数据结构课程设计报告

班级：　　1618104

学号：　　161810225

姓名：　　王烨文

指导老师：　　高　航

目录

一．课设程序分析及样例分析

1. 完成大致情况

本次数据结构课程设计共完成 14 道题，分别是必做八题以及选做 10，11，13，14，16，23 题，总加分值为 12 分。基本都完成了功能实现以及错误输入提示。

2. 进程管理系统

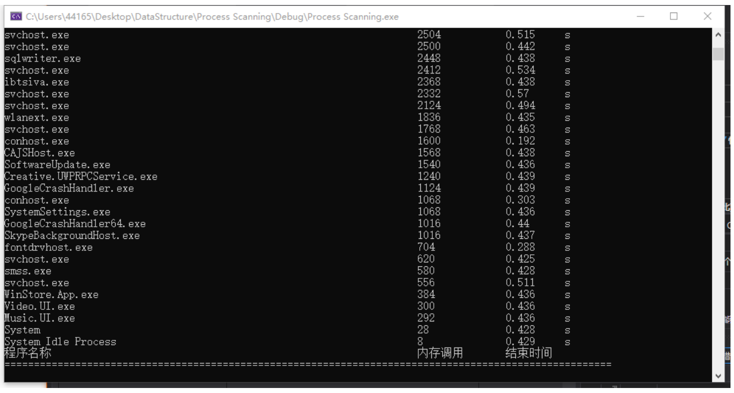
1. 数据结构

   数据结构采用题目要求的单链表与双链表，使用了 map 和 vector 容器进行判断操作。

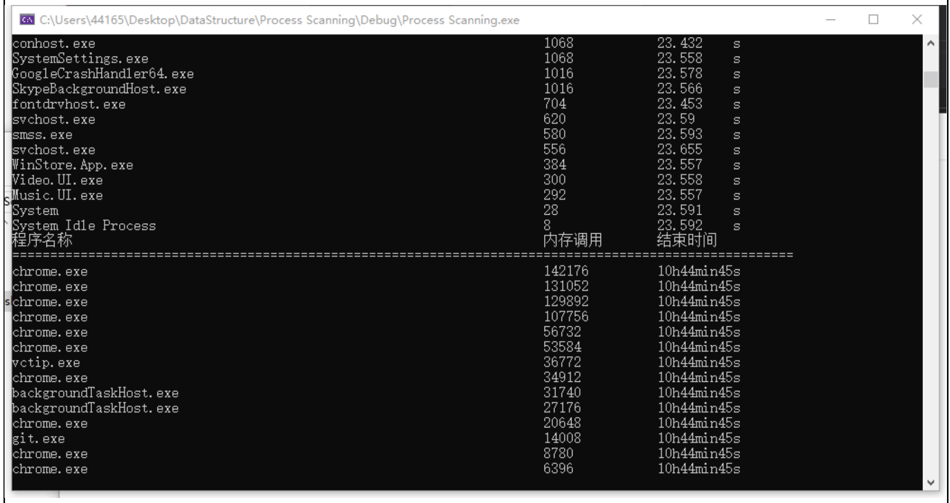
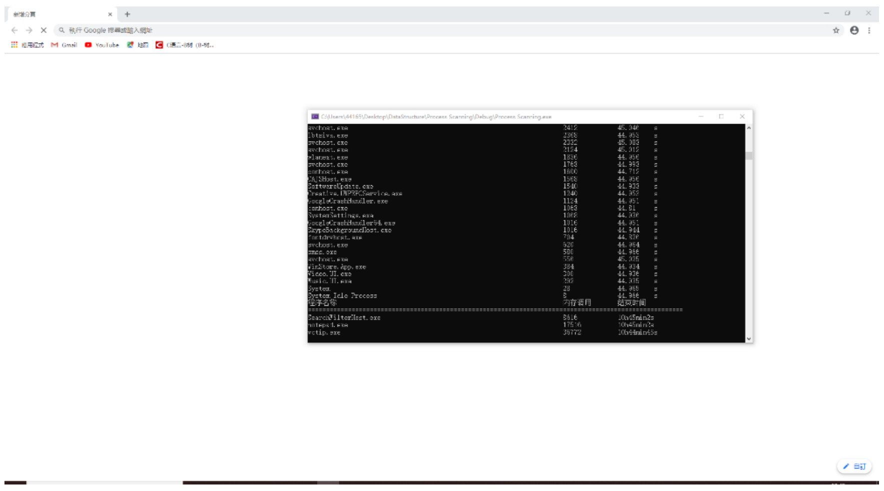
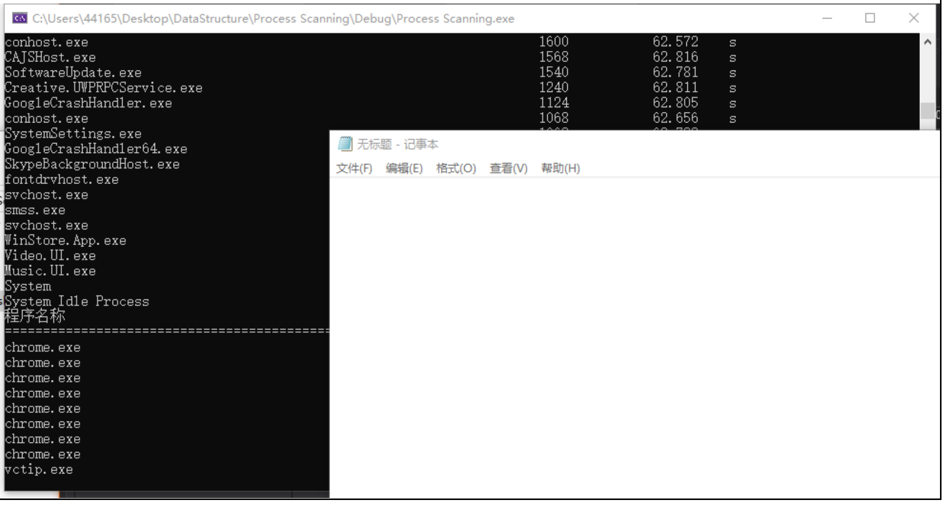
2. 设计思想

   采用了三个链表，两个双链表和一个单链表，一个双链表 L 存放当前进程的所有进程信息，另一个双链表 LL 用于存储所有进程的初始状态即基准组，一个单链表 Lt 用于存储结束进程信息，然后在实际运行过程中，当前进程的信息都需要和基准组进行比对得出运行时间，在显示完当前进程后，将当前组链表与基准组进行对照，如果有新进程出现，基准组记录结束后第一次出现时的初始状态，然后再根据基准组和当前组比对得出结束进程，并且将基准组中相应进程删去，显示结束进程，结束一个时间单位。

3. 测试数据与结果

初始状态	

```
C:\Users\44165\Desktop\DataStructure\Process Scanning\Debug\Process Scanning.exe
svchost.exe                                                     2504          0.515     s
svchost.exe                                                     2500          0.442     s
sqlwriter.exe                                                   2448          0.438     s
svchost.exe                                                     2412          0.534     s
ibtsiva.exe                                                     2368          0.438     s
svchost.exe                                                     2332          0.57      s
svchost.exe                                                     2124          0.494     s
wlanext.exe                                                     1836          0.435     s
svchost.exe                                                     1768          0.463     s
conhost.exe                                                     1600          0.192     s
CAJSHost.exe                                                    1568          0.438     s
SoftwareUpdate.exe                                              1540          0.436     s
Creative.UWPRPCService.exe                                      1240          0.439     s
GoogleCrashHandler.exe                                          1124          0.439     s
conhost.exe                                                     1068          0.303     s
SystemSettings.exe                                              1068          0.436     s
GoogleCrashHandler64.exe                                        1016          0.44      s
SkypeBackgroundHost.exe                                         1016          0.437     s
fontdrvhost.exe                                                 704           0.288     s
svchost.exe                                                     620           0.425     s
smss.exe                                                        580           0.428     s
svchost.exe                                                     556           0.511     s
WinStore.App.exe                                                384           0.436     s
Video.UI.exe                                                    300           0.436     s
Music.UI.exe                                                    292           0.436     s
System                                                          28            0.428     s
System Idle Process                                             8             0.429     s
程序名称                                                        内存调用      结束时间
==========================================================================================================
```

打开 chrome 并关闭	
打开记事本并关闭并再次打开 chrome	
再次打开记事本	

C:\Users\44165\Desktop\DataStructure\Process Scanning\Debug\Process Scanning.exe

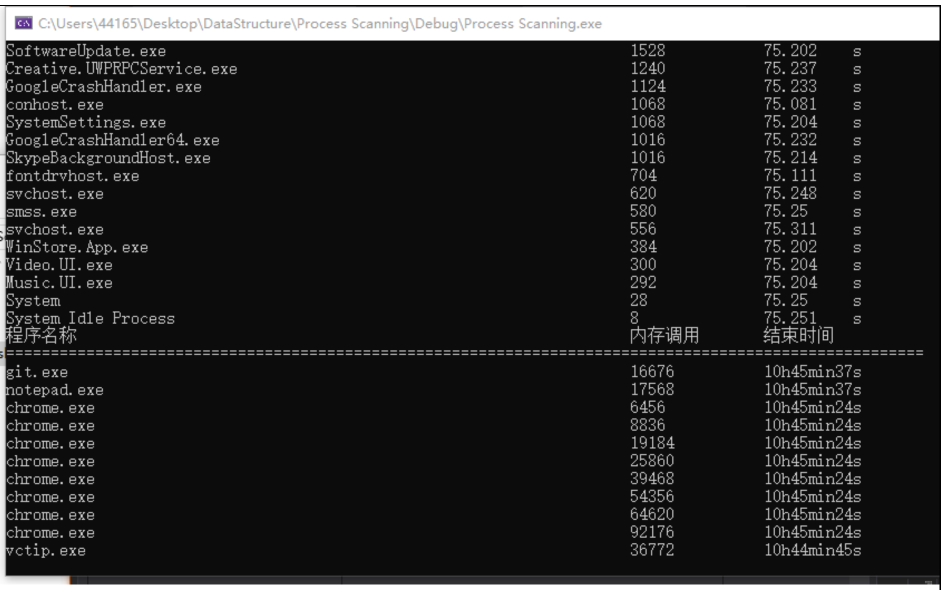
```
conhost.exe                                                        1068          23.432     s
SystemSettings.exe                                                 1068          23.558     s
GoogleCrashHandler64.exe                                           1016          23.578     s
SkypeBackgroundHost.exe                                            1016          23.566     s
fontdrvhost.exe                                                    704           23.453     s
svchost.exe                                                        620           23.59      s
smss.exe                                                           580           23.593     s
svchost.exe                                                        556           23.655     s
WinStore.App.exe                                                   384           23.557     s
Video.UI.exe                                                       300           23.558     s
Music.UI.exe                                                       292           23.557     s
System                                                             28            23.591     s
System Idle Process                                                8             23.592     s
程序名称                                                           内存调用      结束时间
====================================================================================================
chrome.exe                                                         142176        10h44min45s
chrome.exe                                                         131052        10h44min45s
chrome.exe                                                         129892        10h44min45s
chrome.exe                                                         107756        10h44min45s
chrome.exe                                                         56732         10h44min45s
chrome.exe                                                         53584         10h44min45s
vctip.exe                                                          36772         10h44min45s
chrome.exe                                                         34912         10h44min45s
backgroundTaskHost.exe                                             31740         10h44min45s
backgroundTaskHost.exe                                             27176         10h44min45s
chrome.exe                                                         20648         10h44min45s
git.exe                                                            14008         10h44min45s
chrome.exe                                                         8780          10h44min45s
chrome.exe                                                         6396          10h44min45s
```

C:\Users\44165\Desktop\DataStructure\Process Scanning\Debug\Process Scanning.exe

```
svchost.exe                                                        2412          45.046     s
ibtsiva.exe                                                        2368          44.953     s
svchost.exe                                                        2332          45.003     s
svchost.exe                                                        2124          45.012     s
wlanext.exe                                                        1836          44.956     s
svchost.exe                                                        1763          44.993     s
conhost.exe                                                        1600          44.712     s
CAJSHost.exe                                                       1568          44.956     s
SoftwareUpdate.exe                                                 1540          44.933     s
Creative.UWPRPCService.exe                                         1240          44.952     s
GoogleCrashHandler.exe                                             1124          44.951     s
conhost.exe                                                        1068          44.81      s
SystemSettings.exe                                                 1068          44.936     s
GoogleCrashHandler64.exe                                           1016          44.951     s
SkypeBackgroundHost.exe                                            1016          44.944     s
fontdrvhost.exe                                                    704           44.826     s
svchost.exe                                                        620           44.964     s
smss.exe                                                           580           44.966     s
svchost.exe                                                        556           45.025     s
WinStore.App.exe                                                   384           44.934     s
Video.UI.exe                                                       300           44.936     s
Music.UI.exe                                                       292           44.935     s
System                                                             28            44.965     s
System Idle Process                                                8             44.966     s
程序名称                                                           内存调用      结束时间
====================================================================================================
SearchFilterHost.exe                                               8616          10h45min2s
notepad.exe                                                        17516         10h45min2s
vctip.exe                                                          36772         10h44min45s
```

C:\Users\44165\Desktop\DataStructure\Process Scanning\Debug\Process Scanning.exe

```
conhost.exe                                                        1600          62.572     s
CAJSHost.exe                                                       1568          62.816     s
SoftwareUpdate.exe                                                 1540          62.781     s
Creative.UWPRPCService.exe                                         1240          62.811     s
GoogleCrashHandler.exe                                             1124          62.805     s
conhost.exe                                                        1068          62.656     s
SystemSettings.exe
GoogleCrashHandler64.exe
SkypeBackgroundHost.exe
fontdrvhost.exe
svchost.exe
smss.exe
svchost.exe
WinStore.App.exe
Video.UI.exe
Music.UI.exe
System
System Idle Process
程序名称
==========================================
chrome.exe
chrome.exe
chrome.exe
chrome.exe
chrome.exe
chrome.exe
chrome.exe
chrome.exe
vctip.exe
```

再次关闭记事本	

```
C:\Users\44165\Desktop\DataStructure\Process Scanning\Debug\Process Scanning.exe
SoftwareUpdate.exe                                        1528          75.202    s
Creative.UWPRPCService.exe                                1240          75.237    s
GoogleCrashHandler.exe                                    1124          75.233    s
conhost.exe                                               1068          75.081    s
SystemSettings.exe                                        1068          75.204    s
GoogleCrashHandler64.exe                                  1016          75.232    s
SkypeBackgroundHost.exe                                   1016          75.214    s
fontdrvhost.exe                                           704           75.111    s
svchost.exe                                               620           75.248    s
smss.exe                                                  580           75.25     s
svchost.exe                                               556           75.311    s
WinStore.App.exe                                          384           75.202    s
Video.UI.exe                                              300           75.204    s
Music.UI.exe                                              292           75.204    s
System                                                    28            75.25     s
System Idle Process                                       8             75.251    s
程序名称                                                  内存调用      结束时间
=========================================================================================
git.exe                                                   16676         10h45min37s
notepad.exe                                               17568         10h45min37s
chrome.exe                                                6456          10h45min24s
chrome.exe                                                8836          10h45min24s
chrome.exe                                                19184         10h45min24s
chrome.exe                                                25860         10h45min24s
chrome.exe                                                39468         10h45min24s
chrome.exe                                                54356         10h45min24s
chrome.exe                                                64620         10h45min24s
chrome.exe                                                92176         10h45min24s
vctip.exe                                                 36772         10h44min45s
```

可见，第一次初始状态下结束进程没有，而后打开并关闭 chrome 后出现 chrome 以及关闭时间和运行时间，然后打开记事本并关闭并打开 chrome，chrome 在结束进程中被删除，notepad 进程在第一位，按照时间顺序排名，再打开 notepad，notepad 被删除，最后再关闭 notepad，notepad 又出现在前方，git（忽略）。

4. 算法时间复杂度即改进

算法的时间复杂度，遍历算法为 O(n)，circuit 算法即根据链表修改另一个链表信息的时间复杂度为 O(n^2)。改进方法是用一个容器对进程存取，可以是搜索同名进程的时间复杂度由 O(n)降低为 O(logn)，但这失去了本题要求使用链表的思想。

5. 源代码

```
#include<iostream>
#include<stdlib.h>
#include<stdio.h>
#include<vector>
#include<iomanip>
#include<cstring>
#include <windows.h>
#define Ok 1
#define Error 0
using namespace std;
vector<char*> vl;
class task {
public:
 char name[128];
 int memory;
```

```
    SYSTEMTIME start;
    SYSTEMTIME end;
    double time;
    task(char* t) : memory(0)
    {
        char* b = t + 1, *p = t + 1;
        while (*p) {
            if (*p == '"')
            {
                *p = 0;
                strcpy_s(name,128, b);
                break;
            }
            p++;
        }
        for (b = p + 1; p[1] != 'K'; p++)
            if (*p == '"')
                b = p;
        for (char* c = b + 1; c != p; c++)
            if (*c == ',')
                continue;
            else
                memory = 10 * memory + *c - '0';

    }
    bool operator<(task& t) { return this->memory < t.memory; }
};

typedef struct DLNode {
    task run;
    struct DLNode* prior, * next;
}DLNode, * DLinkList;

typedef struct LNode {
    task end;
    struct LNode* next;
}LNode, * LinkList;

int InitDList(DLinkList& L)
{
    L = (DLNode*)malloc(sizeof(struct DLNode));
    if (!L)
        return Error;
    L->next = NULL;
    L->prior = NULL;
    return Ok;
}

int InitList(LinkList& L)
{
    L = (LNode*)malloc(sizeof(struct LNode));
    if (!L)
        return Error;
    L->next = NULL;
    return 1;
}

int ClearDList(DLinkList& L)
{
    free(L->next);
    L->next = NULL;
    return 1;
}

int ClearList(LinkList& L)
{
    free(L->next);
    L->next = NULL;
    return 1;
}

int insertDList(DLinkList& L, DLNode* p)
{
    DLNode* temp = L->next, * temp2 = L->next;
    if (!temp)
    {
        L->next = p;
        p->prior = L;
        return Ok;
    }
    while (temp)
    {
        if (temp->run.memory >= p->run.memory)
            temp = temp->next;
        else
            break;
    }
    if (!temp)
```

```
        {
            while (temp2->next != NULL)
                temp2 = temp2->next;
            temp2->next = p;
            p->prior = temp2;
            return Ok;
        }
        temp->prior->next = p;
        p->next = temp;
        p->prior = temp->prior;
        temp->prior = p;
        return Ok;
    }

    int insertList(LinkList& L, LNode* p)
    {
        LNode* temp = L->next, * temp2=L;
        if (L->next == NULL)
        {
            L->next = p;
            return 1;
        }
        while (temp)
        {
            if (temp->end.end.wHour > p->end.end.wHour || (temp->end.end.wHour == p->end.start.wHour && temp->end.end.wMinute > p->end.end.wMinute) || (temp->end.end.wHour == p->end.end.wHour && temp->end.end.wMinute == p->end.end.wMinute && temp->end.end.wSecond > p->end.end.wMinute))
                temp = temp->next;
            else
                break;
        }
        if (temp == NULL)
        {
            while (temp2->next != NULL)
                temp2 = temp2->next;
            temp2->next = p;
            return 1;
        }
        while (temp2->next != temp)
            temp2 = temp2->next;
        temp2->next = p;
        p->next = temp;
```

```cpp
        return 1;
    }

    void TraverseDList(DLinkList L)
    {
        DLNode* temp = L->next;
        cout << setw(70) << left << "程序名称";
        cout << setw(15) << left << "内存调用";
        cout << setw(10) << left << "持续时间" << endl;
        cout << "===================================================================================================" << endl;
        while (temp)
        {
            cout << setw(70) << left << temp->run.name;
            cout << setw(15) << left << temp->run.memory;
            cout << setw(10) << left << (double)temp->run.time << "s" << endl;
            temp = temp->next;
        }
    }

    void TraverseList(LinkList L)
    {
        LNode* temp = L->next;
        cout << setw(70) << left << "程序名称";
        cout << setw(15) << left << "内存调用";
        cout << setw(10) << left << "结束时间" << endl;
        cout << "=======================================================================================================" << endl;
        while (temp)
        {
            cout << setw(70) << left << temp->end.name;
            cout << setw(15) << left << temp->end.memory;
            cout << left << temp->end.start.wHour << "h" << temp->end.start.wMinute << "min" << temp->end.start.wSecond << "s" << endl;
            temp = temp->next;
        }
    }

    int showpresent(DLinkList& L, DLinkList LL, LinkList& Lt)//显示当前进程
    {
        int flag = 0;
        char buffer[128], cmd[] = "tasklist /FO CSV";
        FILE* pipe = _popen(cmd, "r");
        if (!pipe)
            return Error;
        fgets(buffer, 128, pipe);
        while (!feof(pipe))
        {
            if (fgets(buffer, 128, pipe))
            {
                flag = 0;
                DLNode* p = (DLNode*)malloc(sizeof(struct DLNode));
                DLNode* temp;
                temp = LL->next;
                p->run = task(buffer);
                p->next = NULL;
                p->prior = NULL;
                GetLocalTime(&p->run.start);
                while (temp)
                {
                    if (strcmp(temp->run.name, p->run.name) == 0)
                    {
                        flag = 1;
                        p->run.time = (p->run.start.wMinute - temp->run.start.wMinute) * 60.0 + (p->run.start.wSecond - temp->run.start.wSecond) * 1.0 + (p->run.start.wMilliseconds - temp->run.start.wMilliseconds) * 0.001;
                        break;
                    }
                    temp = temp->next;
                }
                if (flag == 0)
                    p->run.time = 0;
                insertDList(L, p);
            }
        }
        _pclose(pipe);
        TraverseDList(L);
    }

    int showend(LinkList& L, DLinkList LL, DLinkList Lt)//显示结束进程
    {
        int i = 0;
        int flag = 0;
        DLNode* temp = LL->next, * temp2 = Lt->next;
        vector<char*> vnow;
        while (temp2)
        {
            vnow.push_back(temp2->run.name);
            temp2 = temp2->next;
        }
        while (temp)
        {
            int flag = 0;
            vector<char*>::iterator iter;
            iter = vnow.begin();
            while (iter != vnow.end())//LL在Lt中找不到
            {
                if (strcmp(*iter, temp->run.name) == 0)
                {
                    flag = 1;
                    break;
                }
                iter++;
            }
            if (flag == 0)
            {
                LNode* s = (LNode*)malloc(sizeof(struct LNode));
                strcpy_s(s->end.name,128, temp->run.name);
                s->end.memory = temp->run.memory;
                s->end.start = temp->run.start;
                s->next = NULL;
                SYSTEMTIME now;
                GetLocalTime(&now);
                GetLocalTime(&s->end.end);
                s->end.time = (now.wMinute - s->end.start.wMinute) * 60.0 + (now.wSecond - s->end.start.wSecond) * 1.0 + (now.wMilliseconds - s->end.start.wMilliseconds) * 0.001;
                s->end.start = now;
                insertList(L, s);
                vl.push_back(s->end.name);
            }
            temp = temp->next;
        }
        TraverseList(L);
        return 1;
    }

    int InitLL(DLinkList& L)//基准组
```

```
{
    char buffer[128], cmd[] = "tasklist /FO CSV";
    FILE* pipe = _popen(cmd, "r");
    if (!pipe)
        return Error;
    fgets(buffer, 128, pipe);
    while (!feof(pipe))
    {
        if (fgets(buffer, 128, pipe))
        {
            DLNode* p = (DLNode*)malloc(sizeof(struct DLNode));
            p->run = task(buffer);
            p->next = NULL;
            p->prior = NULL;
            GetLocalTime(&p->run.start);
            insertDList(L, p);
        }
    }
    _pclose(pipe);
}

int circuit(LinkList& L, DLinkList L1)//根据present删除结束进程
{
    DLNode* temp = L1->next;
    while (temp)
    {
        vector<char*>::iterator iter;
        iter = vl.begin();
        while (iter != vl.end())
        {
            if (strcmp(*iter, temp->run.name) == 0)
            {
                LNode* temp3 = L->next;
                while (strcmp(*iter, temp3->end.name) != 0)
                {
                    temp3 = temp3->next;
                }
                LNode* temp2 = L;
                while (temp2->next != temp3)
                {
                    temp2 = temp2->next;
                }
                temp2->next = temp3->next;
                vl.erase(iter);
```

```
                    break;
                }
                iter++;
            }
            temp = temp->next;
        }
        return 1;
    }

    int circuitll(DLinkList& pre, DLinkList now)//根据现在校验基准组
    {
        vector<char*> prev;
        DLNode* temp = now->next, * temp2 = pre->next;
        while (temp2)
        {
            prev.push_back(temp2->run.name);
            temp2 = temp2->next;
        }
        temp = now->next, temp2 = pre->next;
        while (temp2)//寻找now中不存在的删掉
        {
            temp = now->next;
            int flag = 0;
            while (temp)
            {
                if (strcmp(temp->run.name, temp2->run.name) == 0)
                {
                    flag = 1;
                    break;
                }
                temp = temp->next;
            }
            if (flag == 0)
            {
                DLNode* temp3 = pre->next;
                while (temp3->next != temp2)
                {
                    temp3 = temp3->next;
                }
                temp2 = temp2->next;
                temp2->prior = temp3;
                temp3->next = temp2;
                continue;
            }
            temp2 = temp2->next;
        }
        temp = now->next, temp2 = pre->next;
        while (temp)//now中存在但LL不存在的添加
        {
            int flag = 0;
            vector<char*>::iterator it;
            it = prev.begin();
            while (it != prev.end())
            {
                if (strcmp(*it, temp->run.name) == 0)//LL中存在的
                {
                    flag = 1;
                    break;
                }
                it++;
            }
            if (flag == 0)
            {
                DLNode* s = (DLNode*)malloc(sizeof(struct DLNode));
                strcpy_s(s->run.name,128, temp->run.name);
                s->run.memory = temp->run.memory;
                s->run.start = temp->run.start;
                s->run.time = temp->run.time;
                insertDList(pre, s);
            }
            temp = temp->next;
        }
        return 1;
    }

    int main()
    {
        DLinkList L, LL;//LL-pre,L-now（LL为基准组
        LinkList L2;//L2-end
        InitDList(L);
        InitDList(LL);
        InitList(L2);
        InitLL(LL);
        long t = time(NULL);
        while (1)
        {
            ClearDList(L);
            showpresent(L, LL, L2);
            circuit(L2, L);
            showend(L2, LL, L);
            circuitll(LL, L);
            cin.ignore();
        }
}
```

3. 算术表达式求值

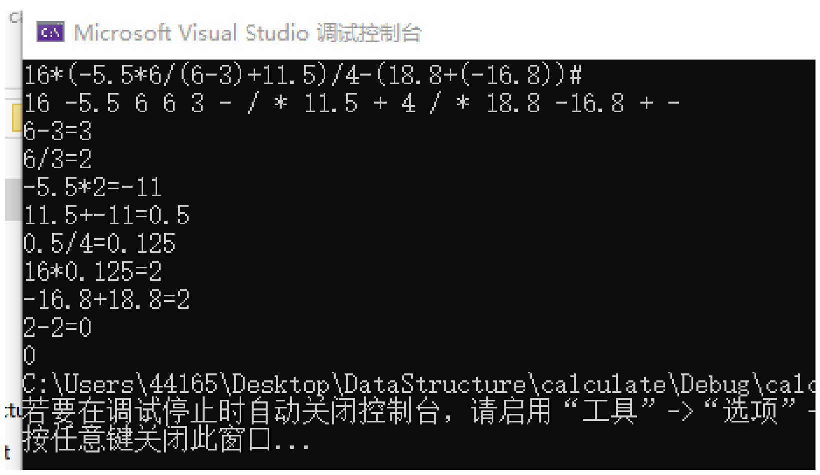
1. 数据结构

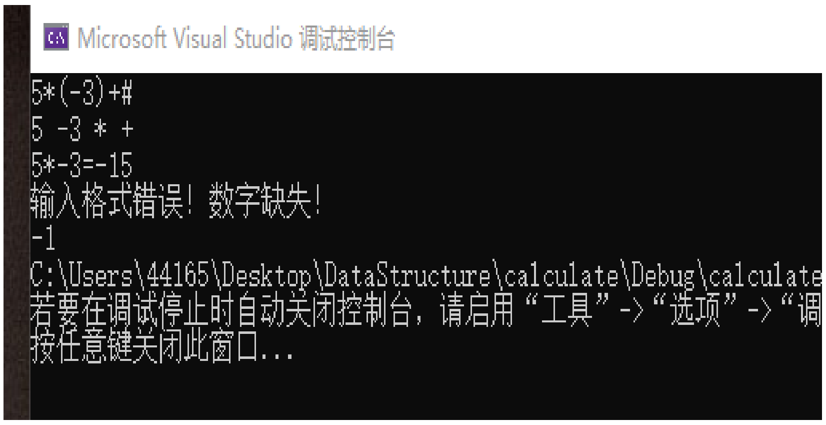
   采用堆栈

2. 设计思想

   先将用户输入的表达式转化为逆波兰表达式，然后再根据逆波兰表达式对输入算式进行求解，小数部分采用了 C++自带函数 atoi，负数部分为自己添加功能，能完成负数的运算。逆波兰表达式转化：遇到数字，则直接放入逆波兰表达式字符串，遇到符号如果没有数字在堆栈中，则先入栈，然后遇到右括号或者#结束标志一并输入到逆波兰字符串中。逆波兰表达式求解即先数字进栈，如果碰到符号则两数字出栈运算后在进栈，因为在逆波兰表达式求解中已经将括号去除。堆栈的变化表现为逐步运算。

3. 测试结果与数据

带负数的
 	
嵌套运算
	 Microsoft Visual Studio 调试控制台
	`16*(-5.5*6/(6-3)+11.5)/4-(18.8+(-16.8))#`
	`16 -5.5 6 6 3 - / * 11.5 + 4 / * 18.8 -16.8 + -`
	`6-3=3`
	`6/3=2`
	`-5.5*2=-11`
	`11.5+-11=0.5`
	`0.5/4=0.125`
	`16*0.125=2`
	`-16.8+18.8=2`
	`2-2=0`
	`0`
	`C:\Users\44165\Desktop\DataStructure\calculate\Debug\calc`
	`若要在调试停止时自动关闭控制台，请启用“工具”->“选项”-`
	`按任意键关闭此窗口...`

非标准格
 	
式输入
	 Microsoft Visual Studio 调试控制台
	`5*(-3)+#`
	`5 -3 * +`
	`5*-3=-15`
	`输入格式错误！数字缺失！`
	`-1`
	`C:\Users\44165\Desktop\DataStructure\calculate\Debug\calculate`
	`若要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调`
	`按任意键关闭此窗口...`

4. 算法时间复杂度及优化

时间复杂度为 O(strlen(str))。优化的话，目前来说已经很好了，也许做个GUI 会更好。

5. 源代码

```
ude<iostream>
#include<stack>
#include<stdlib.h>
#include<stdio.h>
#define stacksize 100
#define add 10
using namespace std;
typedef struct SNode {
  char* base;
  char* top;
  int Stacksize;
  int length;
}SNode, * SqStack;
```

```
stack<double> nu;
char input[1000000], output[1000000];
int InitStack(SqStack& S)
{
 S = (SNode*)malloc(sizeof(SNode));
 S->Stacksize = stacksize;
 S->base = (char*)malloc(sizeof(char) * stacksize);
 S->top = S->base;
 S->length = 0;
 return 1;
}

int Push(SqStack& S, char e)
{
 if (S->top - S->base >= stacksize)
 {
     S->base = (char*)realloc(S->base, sizeof(char) * (stacksize + add));
     S->Stacksize += add;
 }
 *(S->top) = e;
 S->top++;
 S->length++;
 return 1;
}

int Pop(SqStack& S, char& e)
{
 if (S->top == S->base)
     return 0;
 e = *(S->top - 1);
 S->top--;
 S->length--;
 return 1;
}

void Traverse(SqStack S)
{
 cout << "--------------------------------------------------" << endl;
 cout << "OPStack:  ";
 char* p=S->base;
 while (p != S->top)
 {
     cout << *p << " ";
     p++;
 }
 cout << endl;
 cout << "--------------------------------------------------" << endl;
}

int shift(SqStack& op)//转为逆波兰
{
 int flag = 0;
 int i = 0, j = 0, black_white = 1;
 char c;
 while (input[i] != '\0')
 {
     if (input[i] >= '0' && input[i] <= '9' || input[i] == '.')
     {
         if (input[i] == '.')
             flag = 1;
         output[j] = input[i];
         i++;
         j++;
         if (input[i] == '.' && flag == 1)
         {
             cout << "小数点重复出现！格式错误！" << endl;
             return -1;
         }
         if ((input[i]<'0' || input[i]>'9') && input[i] != '.')
         {
             flag = 0;
             output[j] = ' ';
             j++;
         }
     }
     else if (input[i] == '+')
     {
         if (op->length == 0)
         {
             Push(op, input[i]);
         }
         else
         {
             do
             {
                 Pop(op, c);
                     if (c == '(')
                     {
                         Push(op, c);
                         continue;
                     }
                     output[j++] = c;
                     output[j++] = ' ';
                 } while (op->length != 0 && c != '(');
                 Push(op, input[i]);
             }
             i++;
         }
         else if (input[i] == '-' && ((input[i - 1] >= '0' && input[i - 1] <= '9') || input[i - 1] == ')'))
         {
             if (op->length == 0)
             {
                 Push(op, input[i]);
             }
             else
             {
                 do
                 {
                     Pop(op, c);
                     if (c == '(')
                     {
                         Push(op, c);
                         continue;
                     }
                     output[j++] = c;
                     output[j++] = ' ';
                 } while (op->length != 0 && c != '(');
                 Push(op, input[i]);
             }
             i++;
         }
         else if ((input[i] == '-' && i == 0) || (input[i] == '-' && (input[i - 1] == '(' || input[i - 1] == '+' || input[i - 1] == '-' || input[i - 1] == '*' || input[i - 1] == '/')))
         {
             output[j] = input[i];
             j++;
             i++;
         }
         else if (input[i] == '*' || input[i] == '/')
         {
```

```c
            if (op->length == 0)
            {
                Push(op, input[i]);
            }
            else
            {
                do
                {
                    Pop(op, c);
                    if (c == '(')
                    {
                        Push(op, c);
                        continue;
                    }
                    if (c == '+' || c == '-')
                    {
                        Push(op, c);
                        break;
                    }
                    else if (c == '*' || c == '/')
                    {
                        Push(op, c);
                        break;
                    }
                } while (op->length != 0 && c != '(');
                Push(op, input[i]);
            }
            i++;
        }
        else if (input[i] == '(')
        {
            Push(op, input[i]);
            i++;
        }
        else if (input[i] == ')')
        {
            Pop(op, c);
            while (c != '(')
            {
                output[j] = c;
                j++;
                output[j] = ' ';
                j++;
                Pop(op, c);
                }
                i++;
            }
            else if (input[i] == '#')
            {
                if (op->length == 0)
                    break;
                else
                {
                    Pop(op, c);
                    output[j++] = c;
                    output[j++] = ' ';
                }
            }
        }
}

double calculate()//通过逆波兰表达式来计算算式
{
    int i = 0, j = 0;
    double t = 1;
    if (output[0] == '-')
    {
        t = -1;
        i++;
    }
    char num[100];
    double cal = 0;
    while (output[i + 1] != '\0')
    {
        if (output[i] >= '0' && output[i] <= '9' || output[i] == '.')
        {
            num[j++] = output[i];
            i++;
            if (output[i] == ' ')
            {
                if (i >= 100)
                {
                    cout << "数字过大无法计算！" << endl;
                    return -1;
                }
                double a = atof(num);
                nu.push(a * t);
                memset(num, '\0', 100);
```

```cpp
                j = 0;
            }
        }
        else if (output[i] == ' ' && output[i + 1] >= '0' && output[i + 1] <= '9')
        {
            i++;
            t = 1;
            continue;
        }
        else if (output[i] == ' ' && output[i + 1] == '-' && output[i + 2] != ' ')
        {
            i++;
            i++;
            t = -1;
            continue;
        }
        else if (output[i] == ' ' && ((output[i + 1] == '-' && output[i + 2] == ' ') || output[i + 1] == '+' || output[i + 1] == '*' || output[i + 1] == '/' || (output[i + 1] >= '0' && output[i + 1] <= '9')))
        {
            i++;
            continue;
        }
        else if (output[i] == '+')
        {
            double a, b;
            if (nu.empty())
            {
                cout << "输入格式错误！数字缺失！" << endl;
                return -1;
            }
            a = nu.top();
            nu.pop();
            if (nu.empty())
            {
                cout << "输入格式错误！数字缺失！" << endl;
                return -1;
            }
            b = nu.top();
            nu.pop();
            nu.push(a + b);
            i++;
```

```
                cout << a << "+" << b << "=" << a + b << endl;
            }
            else if (output[i] == '/')
            {
                double a, b;
                if (nu.empty())
                {
                    cout << "输入格式错误！数字缺失！" << endl;
                    return -1;
                }
                a = nu.top();
                nu.pop();
                if (nu.empty())
                {
                    cout << "输入格式错误！数字缺失！" << endl;
                    return -1;
                }
                b = nu.top();
                nu.pop();
                if (a == 0)
                {
                    cout << "除数不能为0" << endl;
                    return -1;
                }
                nu.push(b / a);
                i++;
                cout << b << "/" << a << "=" << b / a << endl;
            }
            else if (output[i] == '-')
            {
                double a, b;
                if (nu.empty())
                {
                    cout << "输入格式错误！数字缺失！" << endl;
                    return -1;
                }
                a = nu.top();
                nu.pop();
                if (nu.empty())
                {
                    cout << "输入格式错误！数字缺失！" << endl;
                    return -1;
                }
                b = nu.top();
                nu.pop();
                nu.push(b - a);
                i++;
                cout << b << "-" << a << "=" << b - a << endl;
            }
            else if (output[i] == '*')
            {
                double a, b;
                if (nu.empty())
                {
                    cout << "输入格式错误！数字缺失！" << endl;
                    return -1;
                }
                a = nu.top();
                nu.pop();
                if (nu.empty())
                {
                    cout << "输入格式错误！数字缺失！" << endl;
                    return -1;
                }
                b = nu.top();
                nu.pop();
                nu.push(a * b);
                i++;
                cout << b << "*" << a << "=" << a * b << endl;
            }
        }
        return nu.top();
    }

    int main()
    {
        SqStack op;
        InitStack(op);
        cin >> input;
        shift(op);
        cout << output << endl;
        double cal = calculate();
        cout << cal;
        delete op;
        return 0;
    }
```

4. 公共钥匙盒

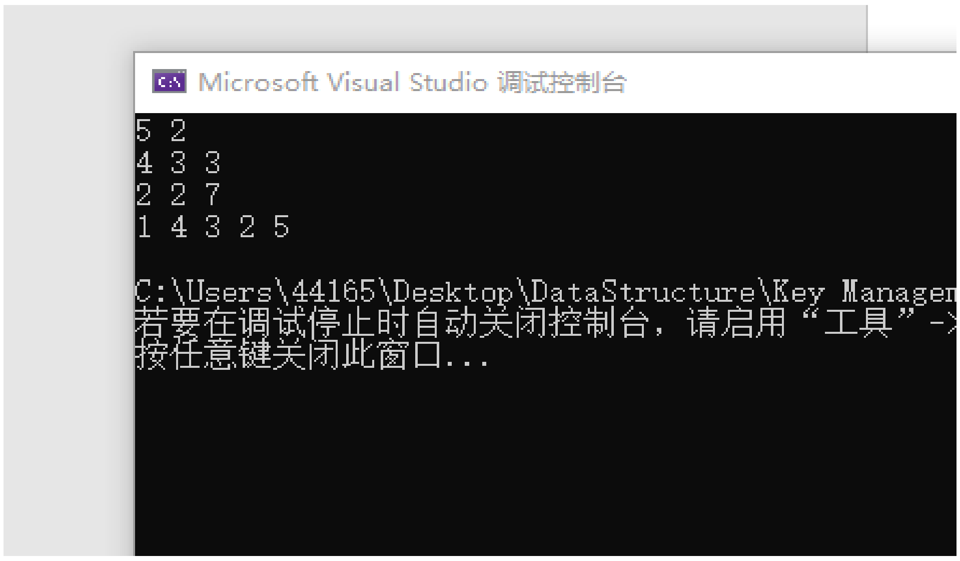
1. 数据结构

   队列。

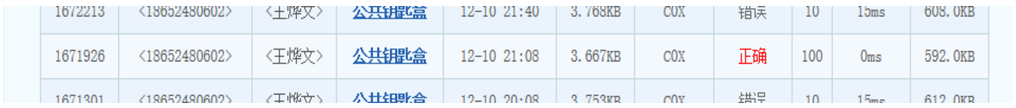
2. 设计思想

   使用双队列，一个队列存放钥匙的取，一个队列存放钥匙的存，然后从存钥匙中出队一个，将它与存钥匙中的进行时间比对，在他结束之前取的钥匙全部取出，然后再存放钥匙。

3. 测试数据及样例



```
5 2
4 3 3
2 2 7
1 4 3 2 5

C:\Users\44165\Desktop\DataStructure\Key Managem
若要在调试停止时自动关闭控制台，请启用 “工具” ->
按任意键关闭此窗口...
```



1672213	<18652480602>	<王烨文>	公共钥匙盒	12-10 21:40	3.768KB	C0X	错误	10	15ms	608.0KB
1671926	<18652480602>	<王烨文>	公共钥匙盒	12-10 21:08	3.667KB	C0X	正确	100	0ms	592.0KB
1671301	<18652480602>	<王烨文>	公共钥匙盒	12-10 20:08	3.753KB	C0X	错误	10	15ms	612.0KB

4. 算法时间复杂度和优化方法

   时间复杂度为 O(n)，n=存放钥匙+取出钥匙，这是一道纯模拟题，如果优化的话，那就是用优先队列进行时间上的比对，然后按照顺序存入优先队列。

5. 源代码

```
#include<iostream>
#include<stdlib.h>
#include<stdio.h>
#include<algorithm>
```

```
using namespace std;
struct key {
 int identity;
 int starttime;
 int endtime;
};

struct keyinfo {
 int identity;
 bool flag;
};

typedef struct QueueNode {
 key keymessage;
 struct QueueNode* next;
}QueueNode,*Queueptr;

typedef struct {
 Queueptr front;
 Queueptr rear;
 int length;
}LinkQueue;
key elem[1001];
keyinfo kp[1001];
int n, time, m;
int InitQueue(LinkQueue& Q)
{
 Q.front = Q.rear = (QueueNode*)malloc(sizeof(struct QueueNode));
 if (!Q.front || !Q.rear)
 {
     cout << "申请空间失败! " << endl;
     return 0;
 }
 Q.front->next = NULL;
 Q.length = 0;
 return 1;
}

int EnQueue(LinkQueue& Q, key e)
{
 QueueNode* p = (QueueNode*)malloc(sizeof(struct QueueNode));
 if (!p)
 {
     cout << "空间申请失败" << endl;
```

```
		return 0;
	}
	p->keymessage = e;
	p->next = NULL;
	Q.rear->next = p;
	Q.rear = p;
	Q.length++;
	return 1;
}

int DeQueue(LinkQueue& Q, key& e)
{
	if (Q.rear == Q.front)
	{
		cout << "队列中不存在元素!" << endl;
		return 0;
	}
	QueueNode* p = (QueueNode*)malloc(sizeof(struct QueueNode));
	if (!p)
	{
		cout << "空间申请失败" << endl;
		return 0;
	}
	p = Q.front->next;
	e = p->keymessage;
	Q.front->next = p->next;
	if (Q.rear == p)
	{
		Q.rear = Q.front;
	}
	free(p);
	Q.length--;
	return 1;
}

int QueueTraverse(LinkQueue Q)
{
	if (Q.rear == Q.front)
	{
		cout << "队列中无元素! " << endl;
		return 0;
	}
	QueueNode* p = (QueueNode*)malloc(sizeof(struct QueueNode));
	if (!p)
```

```cpp
    {
        cout << "空间申请失败" << endl;
        return 0;
    }
    p = Q.front->next;
    while (p != NULL)
    {
        cout << p->keymessage.identity << " " << p->keymessage.starttime << " " << p->keymessage.endtime << endl;
        p = p->next;
    }
    return 1;
}

bool QueueEmpty(LinkQueue Q)
{
    if (Q.front == Q.rear)
        return true;
    return false;
}

bool cmp_starttime(const key &a,const key &b)
{
    if (a.starttime == b.starttime)
        return false;
    return a.starttime < b.starttime;
}

bool cmp_endtime(const key &a,const key &b)
{
    if (a.endtime == b.endtime)
    {
        if (a.identity < b.identity)
            return true;
        return false;
    }
    return a.endtime < b.endtime;
}

void Initkeyinfo()
{
    for (int i = 1; i <= n; i++)
    {
        kp[i].flag = true;
        kp[i].identity = i;
    }
}

void PutKey(key a)
{
    for (int i = 1; i <= n; i++)
    {
        if (kp[i].flag == false)
        {
            kp[i].identity = a.identity;
            kp[i].flag = true;
            i = n;
        }
    }
}

void TakeKey(key a)
{
    for (int i = 1; i <= n; i++)
    {
        if (kp[i].identity == a.identity&&kp[i].flag==true)
        {
            kp[i].flag = false;
            i = n;
        }
    }
}

void Traversekey()
{
    for (int i = 1; i < n; i++)
    {
        cout << kp[i].identity << " ";
    }
    cout << kp[n].identity;
    cout << endl;
}

void keymanage(LinkQueue& take, LinkQueue& back)
{
    int temp = 0;
    key tk, bk;
    DeQueue(take, tk);
    DeQueue(back, bk);
    while ((!QueueEmpty(take) && !QueueEmpty(back))||temp==1)
    {
        if (tk.starttime < bk.endtime)
        {
            TakeKey(tk);
            if (take.length == 0)
                break;
            DeQueue(take, tk);
            if (take.length == 0)
                temp = 1;
        }
        else if (tk.starttime >= bk.endtime)
        {
            PutKey(bk);
            DeQueue(back, bk);
        }
    }
    temp = 0;
    while (!QueueEmpty(back)||temp==1)
    {
        PutKey(bk);
        if (temp == 1)
            break;
        DeQueue(back, bk);
        if (back.length == 0)
            temp = 1;
    }
}

int main()
{
    LinkQueue take,back;
    InitQueue(take);
    InitQueue(back);
    cin >> n>>m;
    Initkeyinfo();
    for(int i=1;i<=m;i++)
    {
        cin >> elem[i].identity >> elem[i].starttime >> time;
        elem[i].endtime = elem[i].starttime + time;
    }
    sort(elem+1, elem + m+1, cmp_starttime);
    for (int i = 1; i <= m; i++)
```

```
        EnQueue(take, elem[i]);
    sort(elem+1, elem + m+1, cmp_endtime);
    for (int i = 1; i <= m; i++)
        EnQueue(back, elem[i]);
    keymanage(take, back);
    Traversekey();
    return 0;
}
```

5. 家谱管理系统

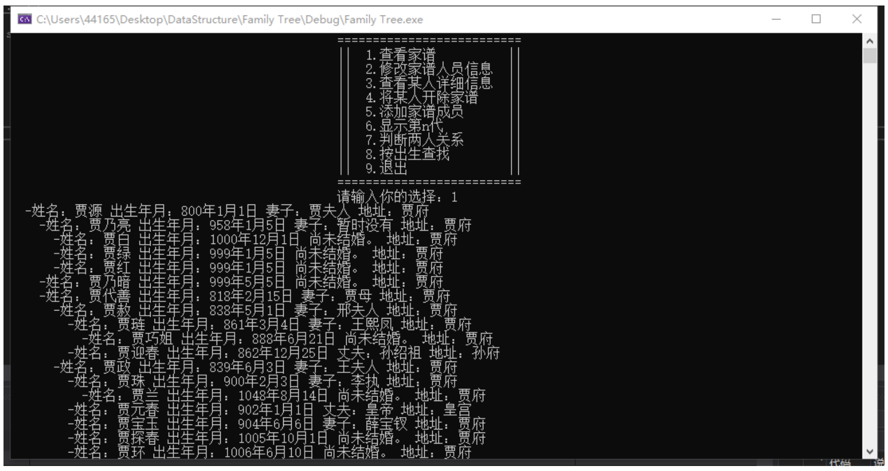
1. 数据结构

孩子兄弟链表。

2. 设计思想

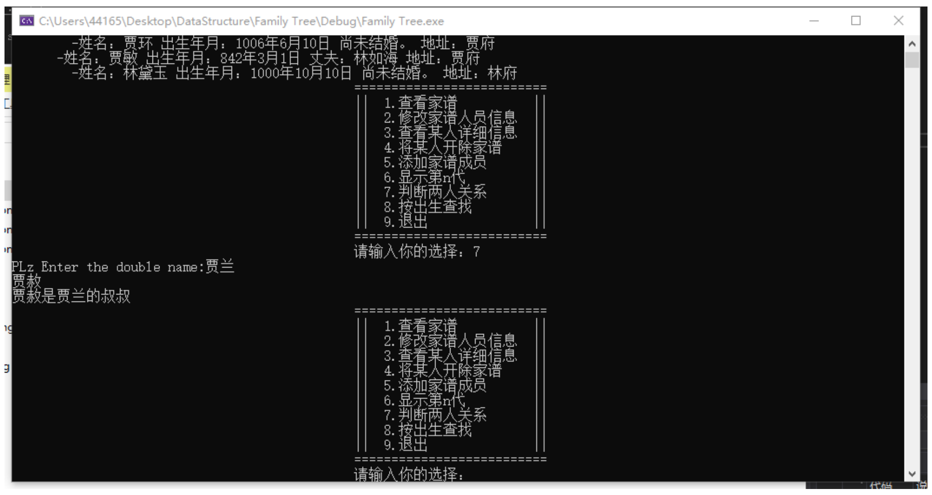
每一个节点存储一个人的信息，信息包括地址性别等个人信息，其实最重要的还是查找函数，我的查找函数同时可以找到人以及他的父亲，找不到人则会返回 0，即查无此人，算法思想是查找时实时更新父亲节点为正在查找的子代的父亲，这样子就免去了再一次的遍历。是整个程序中最有用的函数，在此基础上完成了所有其他功能。

3. 样例输入即测试数据

查看家谱	

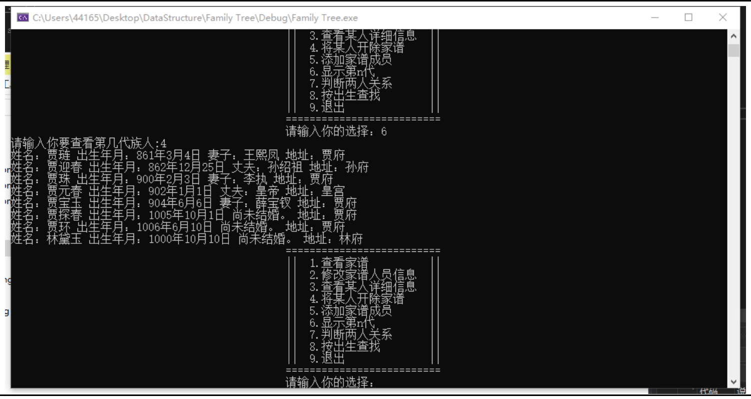
```
C:\Users\44165\Desktop\DataStructure\Family Tree\Debug\Family Tree.exe
                                                =========================
                                                ||   1.查看家谱         ||
                                                ||   2.修改家谱人员信息  ||
                                                ||   3.查看某人详细信息  ||
                                                ||   4.将某人开除家谱    ||
                                                ||   5.添加家谱成员      ||
                                                ||   6.显示第n代         ||
                                                ||   7.判断两人关系      ||
                                                ||   8.按出生查找        ||
                                                ||   9.退出              ||
                                                =========================
                                                请输入你的选择：1
-姓名：贾源 出生年月：800年1月1日 妻子：贾夫人 地址：贾府
  -姓名：贾乃亮 出生年月：958年1月5日 妻子：暂时没有 地址：贾府
    -姓名：贾白 出生年月：1000年12月1日 尚未结婚。 地址：贾府
    -姓名：贾绿 出生年月：999年1月5日 尚未结婚。 地址：贾府
    -姓名：贾红 出生年月：999年1月5日 尚未结婚。 地址：贾府
  -姓名：贾乃暗 出生年月：999年5月5日 尚未结婚。 地址：贾府
  -姓名：贾代善 出生年月：818年2月15日 妻子：贾母 地址：贾府
    -姓名：贾赦 出生年月：838年5月1日 妻子：邢夫人 地址：贾府
      -姓名：贾琏 出生年月：861年3月4日 妻子：王熙凤 地址：贾府
        -姓名：贾巧姐 出生年月：888年6月21日 尚未结婚。 地址：贾府
      -姓名：贾迎春 出生年月：862年12月25日 丈夫：孙绍祖 地址：孙府
    -姓名：贾政 出生年月：839年6月3日 妻子：王夫人 地址：贾府
      -姓名：贾珠 出生年月：900年2月3日 妻子：李执 地址：贾府
        -姓名：贾兰 出生年月：1048年8月14日 尚未结婚。 地址：贾府
      -姓名：贾元春 出生年月：902年1月1日 丈夫：皇帝 地址：皇宫
      -姓名：贾宝玉 出生年月：904年6月6日 妻子：薛宝钗 地址：贾府
      -姓名：贾探春 出生年月：1005年10月1日 尚未结婚。 地址：贾府
      -姓名：贾环 出生年月：1006年6月10日 尚未结婚。 地址：贾府
```

判断关系



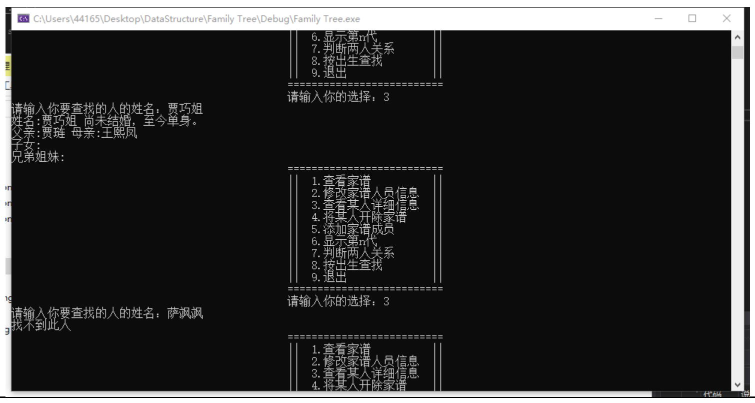
```
C:\Users\44165\Desktop\DataStructure\Family Tree\Debug\Family Tree.exe
       -姓名：贾环 出生年月：1006年6月10日 尚未结婚。 地址：贾府
    -姓名：贾敏 出生年月：842年3月1日 丈夫：林如海 地址：贾府
       -姓名：林黛玉 出生年月：1000年10月10日 尚未结婚。 地址：林府
                                                  =========================
                                                  ||   1.查看家谱           ||
                                                  ||   2.修改家谱人员信息   ||
                                                  ||   3.查看某人详细信息   ||
                                                  ||   4.将某人开除家谱     ||
                                                  ||   5.添加家谱成员       ||
                                                  ||   6.显示第n代          ||
                                                  ||   7.判断两人关系       ||
                                                  ||   8.按出生查找         ||
                                                  ||   9.退出               ||
                                                  =========================
                                                  请输入你的选择：7
PLz Enter the double name:贾兰
贾赦
贾赦是贾兰的叔叔

                                                  =========================
                                                  ||   1.查看家谱           ||
                                                  ||   2.修改家谱人员信息   ||
                                                  ||   3.查看某人详细信息   ||
                                                  ||   4.将某人开除家谱     ||
                                                  ||   5.添加家谱成员       ||
                                                  ||   6.显示第n代          ||
                                                  ||   7.判断两人关系       ||
                                                  ||   8.按出生查找         ||
                                                  ||   9.退出               ||
                                                  =========================
                                                  请输入你的选择：
```

查看代数



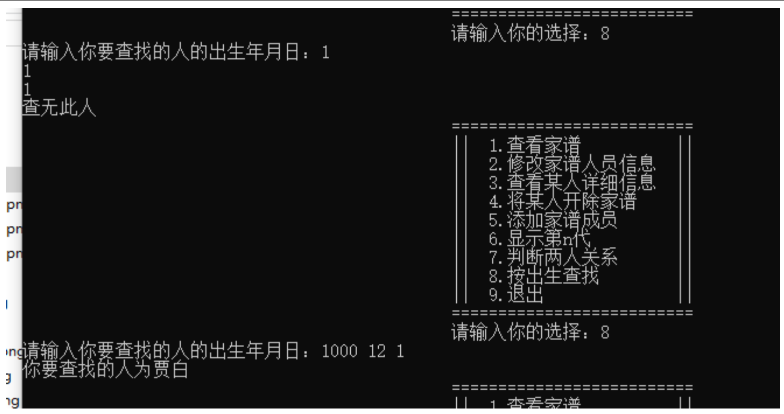
```
C:\Users\44165\Desktop\DataStructure\Family Tree\Debug\Family Tree.exe
                                                  ||   3.查看某人详细信息   ||
                                                  ||   4.将某人开除家谱     ||
                                                  ||   5.添加家谱成员       ||
                                                  ||   6.显示第n代          ||
                                                  ||   7.判断两人关系       ||
                                                  ||   8.按出生查找         ||
                                                  ||   9.退出               ||
                                                  =========================
                                                  请输入你的选择：6
请输入你要查看第几代族人:4
姓名：贾琏 出生年月：861年3月4日 妻子：王熙凤 地址：贾府
姓名：贾迎春 出生年月：862年12月25日 丈夫：孙绍祖 地址：孙府
姓名：贾珠 出生年月：900年2月3日 妻子：李执 地址：贾府
姓名：贾元春 出生年月：902年1月1日 丈夫：皇帝 地址：皇宫
姓名：贾宝玉 出生年月：904年6月6日 妻子：薛宝钗 地址：贾府
姓名：贾探春 出生年月：1005年10月1日 尚未结婚。 地址：贾府
姓名：贾环 出生年月：1006年6月10日 尚未结婚。 地址：贾府
姓名：林黛玉 出生年月：1000年10月10日 尚未结婚。 地址：林府
                                                  =========================
                                                  ||   1.查看家谱           ||
                                                  ||   2.修改家谱人员信息   ||
                                                  ||   3.查看某人详细信息   ||
                                                  ||   4.将某人开除家谱     ||
                                                  ||   5.添加家谱成员       ||
                                                  ||   6.显示第n代          ||
                                                  ||   7.判断两人关系       ||
                                                  ||   8.按出生查找         ||
                                                  ||   9.退出               ||
                                                  =========================
                                                  请输入你的选择：
```

详细信息

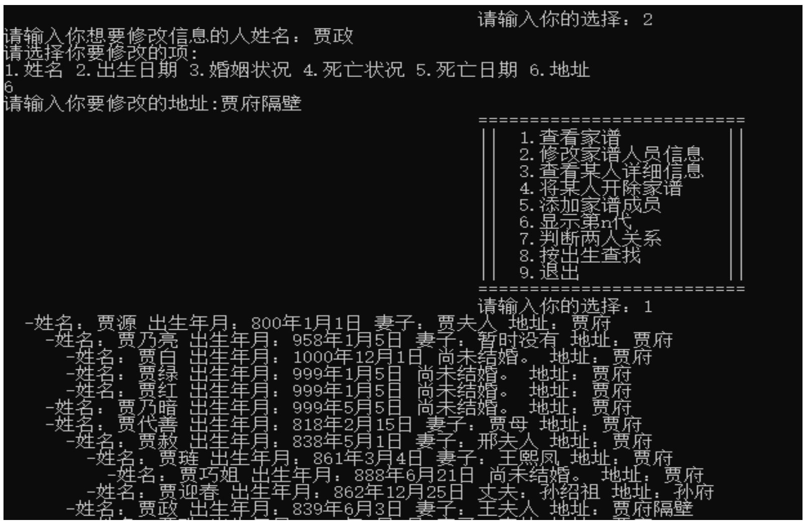
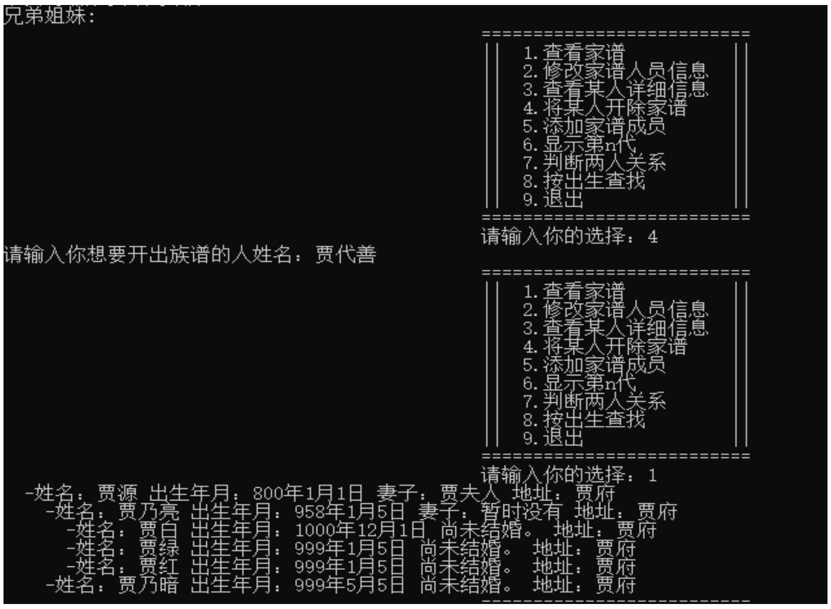
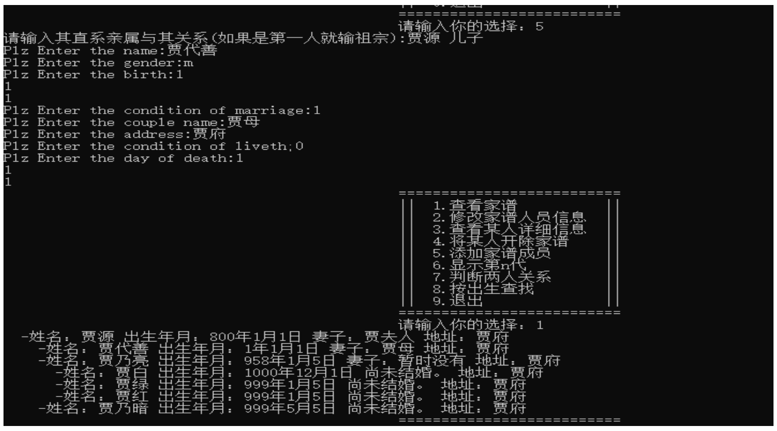


```
C:\Users\44165\Desktop\DataStructure\Family Tree\Debug\Family Tree.exe
                                                  ||   6.显示第n代          ||
                                                  ||   7.判断两人关系       ||
                                                  ||   8.按出生查找         ||
                                                  ||   9.退出               ||
                                                  =========================
                                                  请输入你的选择：3
请输入你要查找的人的姓名：贾巧姐
姓名:贾巧姐 尚未结婚，至今单身。
父亲:贾琏 母亲:王熙凤
子女:
兄弟姐妹:

                                                  =========================
                                                  ||   1.查看家谱           ||
                                                  ||   2.修改家谱人员信息   ||
                                                  ||   3.查看某人详细信息   ||
                                                  ||   4.将某人开除家谱     ||
                                                  ||   5.添加家谱成员       ||
                                                  ||   6.显示第n代          ||
                                                  ||   7.判断两人关系       ||
                                                  ||   8.按出生查找         ||
                                                  ||   9.退出               ||
                                                  =========================
                                                  请输入你的选择：3
请输入你要查找的人的姓名：萨飒飒
找不到此人

                                                  =========================
                                                  ||   1.查看家谱           ||
                                                  ||   2.修改家谱人员信息   ||
                                                  ||   3.查看某人详细信息   ||
                                                  ||   4.将某人开除家谱     ||
```

出生查找



```
                                    =========================
                                    请输入你的选择：8
请输入你要查找的人的出生年月日：1
1
1
查无此人

                                    =========================
                                    ||   1.查看家谱           ||
                                    ||   2.修改家谱人员信息   ||
                                    ||   3.查看某人详细信息   ||
                                    ||   4.将某人开除家谱     ||
                                    ||   5.添加家谱成员       ||
                                    ||   6.显示第n代          ||
                                    ||   7.判断两人关系       ||
                                    ||   8.按出生查找         ||
                                    ||   9.退出               ||
                                    =========================
                                    请输入你的选择：8
请输入你要查找的人的出生年月日：1000 12 1
你要查找的人为贾白

                                    =========================
                                    ||   1.查看家谱           ||
```

修改信息	
删除某人	
添加某人	

```
                                                    请输入你的选择：2
请输入你想要修改信息的人姓名：贾政
请选择你要修改的项:
1.姓名 2.出生日期 3.婚姻状况 4.死亡状况 5.死亡日期 6.地址
6
请输入你要修改的地址:贾府隔壁
                                                    ==========================
                                                    ||   1.查看家谱          ||
                                                    ||   2.修改家谱人员信息  ||
                                                    ||   3.查看某人详细信息  ||
                                                    ||   4.将某人开除家谱    ||
                                                    ||   5.添加家谱成员      ||
                                                    ||   6.显示第n代         ||
                                                    ||   7.判断两人关系      ||
                                                    ||   8.按出生查找        ||
                                                    ||   9.退出              ||
                                                    ==========================
                                                    请输入你的选择：1
  -姓名：贾源 出生年月：800年1月1日 妻子：贾夫人 地址：贾府
    -姓名：贾乃亮 出生年月：958年1月5日 妻子：暂时没有 地址：贾府
      -姓名：贾白 出生年月：1000年12月1日 尚未结婚。 地址：贾府
      -姓名：贾绿 出生年月：999年1月5日 尚未结婚。 地址：贾府
      -姓名：贾红 出生年月：999年1月5日 尚未结婚。 地址：贾府
    -姓名：贾乃暗 出生年月：999年5月5日 尚未结婚。 地址：贾府
    -姓名：贾代善 出生年月：818年2月15日 妻子：贾母 地址：贾府
      -姓名：贾赦 出生年月：838年5月1日 妻子：邢夫人 地址：贾府
        -姓名：贾琏 出生年月：861年3月4日 妻子：王熙凤 地址：贾府
          -姓名：贾巧姐 出生年月：888年6月21日 尚未结婚。 地址：贾府
        -姓名：贾迎春 出生年月：862年12月25日 丈夫：孙绍祖 地址：孙府
      -姓名：贾政 出生年月：839年6月3日 妻子：王夫人 地址：贾府隔壁
```

```
兄弟姐妹:
                                                    ==========================
                                                    ||   1.查看家谱          ||
                                                    ||   2.修改家谱人员信息  ||
                                                    ||   3.查看某人详细信息  ||
                                                    ||   4.将某人开除家谱    ||
                                                    ||   5.添加家谱成员      ||
                                                    ||   6.显示第n代         ||
                                                    ||   7.判断两人关系      ||
                                                    ||   8.按出生查找        ||
                                                    ||   9.退出              ||
                                                    ==========================
                                                    请输入你的选择：4
请输入你想要开出族谱的人姓名：贾代善
                                                    ==========================
                                                    ||   1.查看家谱          ||
                                                    ||   2.修改家谱人员信息  ||
                                                    ||   3.查看某人详细信息  ||
                                                    ||   4.将某人开除家谱    ||
                                                    ||   5.添加家谱成员      ||
                                                    ||   6.显示第n代         ||
                                                    ||   7.判断两人关系      ||
                                                    ||   8.按出生查找        ||
                                                    ||   9.退出              ||
                                                    ==========================
                                                    请输入你的选择：1
  -姓名：贾源 出生年月：800年1月1日 妻子：贾夫人 地址：贾府
    -姓名：贾乃亮 出生年月：958年1月5日 妻子：暂时没有 地址：贾府
      -姓名：贾白 出生年月：1000年12月1日 尚未结婚。 地址：贾府
      -姓名：贾绿 出生年月：999年1月5日 尚未结婚。 地址：贾府
      -姓名：贾红 出生年月：999年1月5日 尚未结婚。 地址：贾府
    -姓名：贾乃暗 出生年月：999年5月5日 尚未结婚。 地址：贾府
                                                    ==========================
```

```
                                                    ==========================
                                                    请输入你的选择：5
请输入其直系亲属与其关系(如果是第一人就输祖宗):贾源 儿子
Plz Enter the name:贾代善
Plz Enter the gender:m
Plz Enter the birth:1
1
1
Plz Enter the condition of marriage:1
Plz Enter the couple name:贾母
Plz Enter the address:贾府
Plz Enter the condition of liveth;0
Plz Enter the day of death:1
1
1
                                                    ==========================
                                                    ||   1.查看家谱          ||
                                                    ||   2.修改家谱人员信息  ||
                                                    ||   3.查看某人详细信息  ||
                                                    ||   4.将某人开除家谱    ||
                                                    ||   5.添加家谱成员      ||
                                                    ||   6.显示第n代         ||
                                                    ||   7.判断两人关系      ||
                                                    ||   8.按出生查找        ||
                                                    ||   9.退出              ||
                                                    ==========================
                                                    请输入你的选择：1
  -姓名：贾源 出生年月：800年1月1日 妻子：贾夫人 地址：贾府
    -姓名：贾代善 出生年月：1年1月1日 妻子：贾母 地址：贾府
    -姓名：贾乃亮 出生年月：958年1月5日 妻子：暂时没有 地址：贾府
      -姓名：贾白 出生年月：1000年12月1日 尚未结婚。 地址：贾府
      -姓名：贾绿 出生年月：999年1月5日 尚未结婚。 地址：贾府
      -姓名：贾红 出生年月：999年1月5日 尚未结婚。 地址：贾府
    -姓名：贾乃暗 出生年月：999年5月5日 尚未结婚。 地址：贾府
                                                    ==========================
```

4. 算法时间复杂度即优化

时间复杂度为 O(n)，大多数为查找，优化可以进行图形表示，树本身的优化可以多注意结点空间释放，以及指针空间申请的规范。

5. 源代码

```
#include<iostream>
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#include<queue>
#include<stack>
#include<fstream>
#include<iomanip>
using namespace std;
struct people {
 char gender = 'f';
 char name[20]=" ";
 int born_year=0;
 int born_month=0;
 int born_day=0;
 bool marriage=0;
 char address[20]=" ";
 char couple[20]=" ";
 bool alive=1;
 int die_year=0;
 int die_month=0;
 int die_day=0;
};
typedef struct CSNode {
 people data;
 int number;
 struct CSNode* firstchild, * nextsibling;
}CSNode,*CSTree;
```

```
queue<CSNode*> cstree;
int InitCSTree(CSTree& T)
{
 T = (CSNode*)malloc(sizeof(struct CSNode));
 if (!T)
 {
        cout << "空间内存申请失败！ " << endl;
        return 0;
 }
 T->number = 0;
 return 1;
}
CSNode* parent = (CSNode*)malloc(sizeof(struct CSNode));
CSNode* silding = (CSNode*)malloc(sizeof(struct CSNode));
CSNode* temp = (CSNode*)malloc(sizeof(struct CSNode));
void Createman(people& man)
{
 cout << "Plz Enter the name:";
 cin >> man.name;
 cout << "Plz Enter the gender:";
 cin >> man.gender;
 cout << "Plz Enter the birth:";
 cin >> man.born_year >> man.born_month >> man.born_day;
 cout << "Plz Enter the condition of marriage:";
 cin >> man.marriage;
 if (man.marriage == 1)
 {
        cout << "Plz Enter the couple name:";
        cin >> man.couple;
 }
 cout << "Plz Enter the address:";
 cin >> man.address;
 cout << "Plz Enter the condition of liveth;";
 cin >> man.alive;
 if (man.alive == 1)
 {
 	man.die_year = 0;
 	man.die_month = 0;
 	man.die_day = 0;
 }
 else if (man.alive == 0)
 {
 	cout << "Plz Enter the day of death:";
 	cin >> man.die_year >> man.die_month >> man.die_day;
 }
}

void Createman(people& man,fstream &fp)
{
 fp >> man.name;
 fp >> man.gender;
 fp >> man.born_year >> man.born_month >> man.born_day;
 fp >> man.marriage;
 if (man.marriage == 1)
 {
 	fp >> man.couple;
 }
 fp >> man.address;
 fp >> man.alive;
 if (man.alive == 1)
 {
        man.die_year = 0;
        man.die_month = 0;
        man.die_day = 0;
 }
 else if (man.alive == 0)
 {
        fp >> man.die_year >> man.die_month >> man.die_day;
 }
}

int Find(CSTree T, CSTree& parent, CSTree& silbing, char *name)//直接找父亲和兄弟的节点定位，自
我感觉最巧妙的函数
{
 int flag = 0;
 temp = T;
 parent = T;
 if (strcmp(temp->data.name,name)==0)
 {
        silbing = temp;
        return 1;
 }
 cstree.push(temp);
 while (!cstree.empty()&&flag==0)
 {
        temp = cstree.front();
        cstree.pop();
        if (temp->firstchild)
        {
               parent = temp;
```

```
                        temp = temp->firstchild;
                        while (temp)
                        {
                                if (strcmp(temp->data.name, name) == 0)
                                {
                                        flag = 1;
                                        silbing = temp;
                                        break;
                                }
                                cstree.push(temp);
                                temp = temp->nextsibling;
                        }
                }
        }
        while (!cstree.empty())
        {
                cstree.pop();
        }
        return flag;
}

void Showman(people man)
{
        cout << "姓名：" << man.name << " ";
        cout << "出生年月：" << man.born_year << "年" << man.born_month << "月" << man.born_day
<< "日  ";
        if (man.marriage == 1)
        {
                if (man.gender == 'f')
                {
```

```
			cout << "丈夫：" << man.couple<<" ";
		}
		else
		{
			cout << "妻子：" << man.couple<<" ";
		}
	}
	else if (man.marriage == 0)
	{
		cout << "尚未结婚。";
	}
	cout << "地址：" << man.address << endl;
}

int Insertman(CSTree& T)
{
	char name[20],relation[20];
	people man;
	cout << "请输入其直系亲属与其关系(如果是第一人就输祖宗):";
	cin >> name >> relation;
	if (strcmp(relation, "祖宗") == 0)
	{
		if (T->number==0)
		{
			Createman(man);
			T->data = man;
			T->firstchild = NULL;
			T->nextsibling = NULL;
			T->number++;
			return 1;
```

```
        }
        else if (T->number != 0)
        {
            Createman(man);
            CSNode* p = (CSNode*)malloc(sizeof(struct CSNode));
            if (p == NULL)
            {
                cout << "申请失败！ " << endl;
                return -1;
            }
            p->data = man;
            p->nextsibling = NULL;
            p->firstchild = T;
            T = p;
            T->number++;
            return 1;
        }
    }
    else if (strcmp(relation, "哥哥") == 0 || strcmp(relation, "弟弟") == 0 || strcmp(relation, "姐姐") == 0 ||
strcmp(relation, "妹妹") == 0 || strcmp(relation, "儿子") == 0 || strcmp(relation, "女儿") == 0)
    {
        if (T->number == 0)
        {
            cout << "家谱中没有人" << endl;
            return -1;
        }
        else if (T->number == 1&&(strcmp(relation, "哥哥") == 0 || strcmp(relation, "弟弟") == 0))
        {
            cout << "祖先列暂不设置多人" << endl;
            return -1;
```

```
}
else
{
    Createman(man);
    CSNode* p = (CSNode*)malloc(sizeof(struct CSNode));
    if (p == NULL)
    {
        cout << "申请失败！ " << endl;
        return -1;
    }
    p->data = man;
    p->firstchild = NULL;
    p->nextsibling = NULL;
    if (strcmp(relation, "弟弟") == 0 || strcmp(relation, "妹妹") == 0)
    {
        int t=Find(T, parent, silding, name);
        if (t == 0)
        {
            cout << "找不到此人" << endl;
            return -1;
        }
        p->nextsibling = silding->nextsibling;
        silding->nextsibling = p;
        p->firstchild = NULL;
        T->number++;
    }
    else if (strcmp(relation, "姐姐") == 0 || strcmp(relation, "哥哥") == 0)
    {
        int t=Find(T, parent, silding, name);
        if (t == 0)
        {
            cout << "找不到此人" << endl;
            return -1;
        }
        CSNode* temp = (CSNode*)malloc(sizeof(struct CSNode));
        temp = parent->firstchild;
        if (temp == silding)
        {
            p->nextsibling = temp;
            parent->firstchild = p;
            T->number++;
        }
        else
        {
            while (temp->nextsibling != silding)
            {
                temp = temp->nextsibling;
            }
            temp->nextsibling = p;
            p->nextsibling = silding;
            T->number++;
        }
    }
    else if (strcmp(relation, "儿子") == 0 || strcmp(relation, "女儿") == 0)
    {
        if (parent == NULL)
        {
            cout << "申请失败" << endl;
            return -1;
        }
```

```
                int t=Find(T, silding, parent,name);
                if (t == 0)
                {
                    cout << "找不到此人" << endl;
                    return -1;
                }
                if (!parent->firstchild)
                {
                    parent->firstchild = p;
                }
                else
                {
                    p->nextsibling = parent->firstchild;
                    parent->firstchild = p;
                }
                T->number++;
            }
            return 1;
        }
    }
    return 1;
}

void TraverseFT(CSTree T)
{
    ofstream fout("output.txt");
    int i = 0;
    if (T == NULL)
        return;
    temp = T;
```

```
stack<CSNode*> cstree1;
while (!cstree1.empty() || temp)
{
        while (temp)
        {
                for (int j = 0; j <= i; j++)
                {
                        cout << "    ";
                        fout << "    ";
                }
                cout << "-";
                fout << "-";
                Showman(temp->data);
                fout << "姓名：" << temp->data.name << " ";
                fout << "出生年月：" << temp->data.born_year << "年" << temp->data.born_month <<
"月" << temp->data.born_day << "日  ";
                if (temp->data.marriage == 1)
                {
                        if (temp->data.gender == 'f')
                        {
                                fout << "丈夫：" << temp->data.couple << " ";
                        }
                        else
                        {
                                fout << "妻子：" << temp->data.couple << " ";
                        }
                }
                else if (temp->data.marriage == 0)
                {
                        fout << "尚未结婚。  ";
```

```
				}
				fout << "地址：" << temp->data.address << endl;
				fout << endl;
				cstree1.push(temp);
				temp = temp->firstchild;
				i++;
			}
			if (!cstree1.empty())
			{
				temp = cstree1.top();
				cstree1.pop();
				temp = temp->nextsibling;
				i--;
			}
	}
}

int CreateCSTree(CSTree& T)
{
	fstream fp;
	fp.open("1.txt",ios::in);
	int n;
	fp >> n;
	while (n--)
	{
			char name[20], relation[20];
			fp >> name >> relation;
			if (strcmp(relation, "祖宗")==0)
			{
				if (T->number == 0)
```

```
                {
                        Createman(T->data,fp);
                        T->firstchild = NULL;
                        T->nextsibling = NULL;
                        T->number++;
                }
                else if (T->number != 0)
                {

                        CSNode* p = (CSNode*)malloc(sizeof(struct CSNode));
                        if (p == NULL)
                        {
                                cout << "申请失败！" << endl;
                                fp.close();
                                return -1;
                        }
                        Createman(p->data,fp);
                        p->nextsibling = NULL;
                        p->firstchild = T;
                        T = p;
                        T->number++;
                }
        }
        else if (strcmp(relation,"哥哥")==0|| strcmp(relation, "弟弟") == 0 || strcmp(relation, "姐姐") == 0
|| strcmp(relation, "妹妹") == 0 || strcmp(relation, "儿子") == 0 || strcmp(relation, "女儿") == 0)
        {
                if (T->number == 0)
                {
                        cout << "家谱中没有人" << endl;
                        return -1;
```

```
		}
		else if (T->number == 1 && (strcmp(relation, "哥哥") == 0 || strcmp(relation, "弟弟") ==
0))
		{
			cout << "祖先列暂不设置多人" << endl;
			return -1;
		}
		else
		{
			CSNode* p = (CSNode*)malloc(sizeof(struct CSNode));
			if (p == NULL)
			{
				cout << "申请失败！" << endl;
				return -1;
			}
			Createman(p->data,fp);
			p->firstchild = NULL;
			p->nextsibling = NULL;
			if (strcmp(relation, "弟弟") == 0 || strcmp(relation, "妹妹") == 0)
			{
				int t=Find(T, parent, silding, name);
				if (t == 0)
				{
					cout << "找不到此人" << endl;
					return -1;
				}
				p->nextsibling = silding->nextsibling;
				silding->nextsibling = p;
				p->firstchild = NULL;
				T->number++;
```

```
}
else if (strcmp(relation, "姐姐") == 0 ||strcmp(relation, "哥哥") == 0)
{
    int t=Find(T, parent, silding, name);
    if (t == 0)
    {
        cout << "找不到此人" << endl;
        return -1;
    }
    CSNode* temp = (CSNode*)malloc(sizeof(struct CSNode));
    temp = parent->firstchild;
    if (temp == silding)
    {
        p->nextsibling = temp;
        parent->firstchild = p;
        T->number++;
    }
    else
    {
        while (temp->nextsibling != silding)
        {
            temp = temp->nextsibling;
        }
        temp->nextsibling = p;
        p->nextsibling = silding;
        T->number++;
    }
}
else if (strcmp(relation, "儿子") == 0 || strcmp(relation, "女儿") == 0)
{
```

```
fp.close();

return 1;

}

                if (parent == NULL)

                {

                        cout << "申请失败" << endl;

                        fp.close();

                        return -1;

                }

                int t=Find(T, silding, parent, name);

                if (t == 0)

                {

                        cout << "找不到此人" << endl;

                        return -1;

                }

                if (!parent->firstchild)

                {

                        parent->firstchild = p;

                }

                else

                {

                        p->nextsibling = parent->firstchild;

                        parent->firstchild = p;

                }

                T->number++;

            }

        }

    }

}

fp.close();

return 1;

}
```

```
void Deleteman(CSTree& T)
{
 char name[20];
 cout << "请输入你想要开出族谱的人姓名：";
 cin >> name;
 int t=Find(T, parent, silding, name);
 if (t == 0)
 {
        cout << "找不到此人" << endl;
        return;
 }
 if (silding == T)
 {
        cout << "祖先不可变" << endl;
 }
 else
 {
        if (parent->firstchild == silding)
        {
               parent->firstchild = silding->nextsibling;
               free(silding);
        }
        else
        {
               parent = parent->firstchild;
               while (parent->nextsibling != silding)
               {
                      parent = parent->nextsibling;
               }
               parent->nextsibling = silding->nextsibling;
```

```
			free(silding);
		}
	}
}

void modifyinfo(CSTree& T)
{
	char name[20];
	cout << "请输入你想要修改信息的人姓名：";
	cin >> name;
	int t=Find(T, parent, silding, name);
	if (t == 0)
	{
		cout << "找不到此人" << endl;
		return;
	}
	cout << "请选择你要修改的项:" << endl;
	cout << "1.姓名 " << "2.出生日期 " << "3.婚姻状况 " << "4.死亡状况 " << "5.死亡日期 " << "6.地
址" << endl;
	int choice;
	cin >> choice;
	if (choice == 1)
	{
		cout << "请输入你要修改的姓名:";
		cin >> silding->data.name;
	}
	if (choice == 2)
	{
		cout << "请输入你要修改的生日:";
		cin >> silding->data.born_year>>silding->data.born_month>>silding->data.born_day;
 }
 if (choice == 3)
 {
     cout << "请输入你要修改的婚姻:";
     cin >> silding->data.marriage;
     if (silding->data.marriage == 1)
     {
         cout << "Plz Enter the couple name:";
         cin >> silding->data.couple;
     }
 }
 if (choice == 4)
 {
     cout << "请输入你要修改的死亡状况:";
     cin >> silding->data.alive;
 }
 if (choice == 5)
 {
     cout << "请输入你要修改的死亡日期:";
     cin >> silding->data.die_year>>silding->data.born_month>>silding->data.die_day;
 }
 if (choice == 6)
 {
     cout << "请输入你要修改的地址:";
     cin >> silding->data.address;
 }
}

int Findman(CSTree T)
{
```

```
char name[20];
cout << "请输入你要查找的人的姓名：";
cin >> name;
int t=Find(T, parent, silding, name);
if (t == 0)
{
        cout << "找不到此人" << endl;
        return -1;
}
cout << "姓名:" << silding->data.name;
if (silding->data.marriage == 1)
{
        if (silding->data.gender == 'f')
        {
                cout << "  丈夫:" << silding->data.couple<<endl;
        }
        else
        {
                cout << "  妻子:" << silding->data.couple << endl;
        }
}
else if (silding->data.marriage == 0)
{
        cout << "  尚未结婚，至今单身。" << endl;
}
cout<< "父亲:" << parent->data.name <<"  母亲:"<<parent->data.couple<< endl;
silding = silding->firstchild;
cout << "子女:";
while (silding)
{
        cout << silding->data.name << " ";
        silding = silding->nextsibling;
 }
 cout << endl;
 parent = parent->firstchild;
 cout << "兄弟姐妹:";
 while (parent)
 {
        if (parent->data.name==name)
        {
               cout << parent->data.name << " ";
        }
        parent = parent->nextsibling;
 }
 cout << endl;
 return 1;
}

int Findbirth(CSTree T)
{
 int year, month, day;
 cout << "请输入你要查找的人的出生年月日：";
 cin >> year>>month>>day;
 temp = T;
 cstree.push(temp);
 while (!cstree.empty())
 {
        temp = cstree.front();
        if (temp->data.born_year == year && temp->data.born_month == month &&
temp->data.born_day == day)
```

```
		{
			cout << "你要查找的人为" << temp->data.name << endl;
			while (!cstree.empty())
				cstree.pop();
			return 1;
		}
		cstree.pop();
		if (temp->firstchild)
		{
			temp = temp->firstchild;
			while (temp)
			{
				cstree.push(temp);
				temp = temp->nextsibling;
			}
		}
	}
	cout << "查无此人" << endl;
	return 0;
}

int FindNdynasty(CSTree T)
{
	queue<CSNode*> cstree1, cstree2;
	int n;
	cout << "请输入你要查看第几代族人:";
	cin >> n;
	int i = 1;
	temp = T;
	cstree1.push(temp);
```

```
while (!cstree1.empty() || !cstree2.empty())
{
    if (i == n)
    {
        break;
    }
    if (i % 2 == 1)
    {
        while (!cstree1.empty())
        {
            temp = cstree1.front();
            cstree1.pop();
            temp = temp->firstchild;
            while (temp)
            {
                cstree2.push(temp);
                temp = temp->nextsibling;
            }
        }
        i++;
    }
    else if (i % 2 == 0)
    {
        while (!cstree2.empty())
        {
            temp = cstree2.front();
            cstree2.pop();
            temp = temp->firstchild;
            while (temp)
            {
```

```
					cstree1.push(temp);
					temp = temp->nextsibling;
				}
			}
			i++;
		}
	}
	if (i % 2 == 0)
	{
		while (!cstree2.empty())
		{
			temp = cstree2.front();
			cstree2.pop();
			Showman(temp->data);
		}
	}
	else
	{
		while (!cstree1.empty())
		{
			temp = cstree1.front();
			cstree1.pop();
			Showman(temp->data);
		}
	}
	return 1;
}

int Show(CSTree T)
{
```

```
char name1[20], name2[20];
cout << "PLz Enter the double name:";
cin >> name1 >> name2;
CSNode* parent1 = (CSNode*)malloc(sizeof(struct CSNode));
CSNode* silding1 = (CSNode*)malloc(sizeof(struct CSNode));
CSNode* parent2 = (CSNode*)malloc(sizeof(struct CSNode));
CSNode* silding2 = (CSNode*)malloc(sizeof(struct CSNode));
int t1=Find(T, parent1, silding1, name1);
int t2=Find(T, parent2, silding2, name2);
if (t1 == 0||t2==0)
{
    cout << "找不到此人" << endl;
    return -1;
}
if (strcmp(parent1->data.name, name2) == 0)
{
    if (parent1->data.gender == 'f')
    {
        cout << name2 << "是" << name1 << "的母亲" << endl;
    }
    else if (parent1->data.gender == 'm')
    {
        cout << name2 << "是" << name1 << "的父亲" << endl;
    }
    return 1;
}
if (strcmp(parent2->data.name, name1) == 0)
{
    if (parent2->data.gender == 'f')
    {
            cout << name1 << "是" << name2 << "的母亲" << endl;
        }
        else if (parent2->data.gender == 'm')
        {
            cout << name1 << "是" << name2 << "的父亲" << endl;
        }
        return 1;
    }
    else
    {
        if (strcmp(parent1->data.name, parent2->data.name) == 0)
        {
            cout << name2 << "是" << name1 << "的兄弟姐妹" << endl;
            return 1;
        }
        Find(T, parent1, silding1, name1);
        Find(T, parent2, silding2, parent1->data.name);
        if (strcmp(parent2->data.name, name2) == 0)
        {
            if (parent2->data.gender == 'f')
            {
                cout << name2 << "是" << name1 << "的奶奶" << endl;
            }
            else if (parent2->data.gender == 'm')
            {
                cout << name2 << "是" << name1 << "的爷爷" << endl;
            }
            return 1;
        }
        Find(T, parent1, silding1, name2);
```

```
Find(T, parent2, silding2, parent1->data.name);
if (strcmp(parent2->data.name, name1) == 0)
{
    if (parent2->data.gender == 'f')
    {
        cout << name1 << "是" << name2 << "的奶奶" << endl;
    }
    else if (parent2->data.gender == 'm')
    {
        cout << name1 << "是" << name2 << "的爷爷" << endl;
    }
    return 1;
}
Find(T, parent1, silding1, name1);
Find(T, parent2, silding2, name2);
Find(T, parent, silding, parent1->data.name);
temp = parent->firstchild;
while (temp)
{
    if (strcmp(temp->data.name, parent2->data.name) == 0)
    {
        cout << name1 << "与" << name2 << "为堂兄弟姐妹" << endl;
        return 1;
    }
    temp = temp->nextsibling;
}
Find(T, parent, silding, name1);
Find(T, parent, silding, parent->data.name);
Find(T, parent, silding, parent->data.name);
Find(T, parent, silding, parent->data.name);
```

```
silding = parent->firstchild;
while (silding)
{
    temp = silding->firstchild;
    while (temp)
    {
        if (strcmp(temp->data.name, name2) == 0)
        {
            if (temp->data.gender == 'f')
            {
                cout << name2 << "是" << name1 << "的阿姨" << endl;
            }
            else
            {
                cout << name2 << "是" << name1 << "的叔叔" << endl;
            }
            return 1;
        }
        temp = temp->nextsibling;
    }
    silding = silding->nextsibling;
}
Find(T, parent, silding, name2);
Find(T, parent, silding, parent->data.name);
Find(T, parent, silding, parent->data.name);
Find(T, parent, silding, parent->data.name);
silding = parent->firstchild;
while (silding)
{
    temp = silding->firstchild;
```

```
                while (temp)
                {
                        if (strcmp(temp->data.name, name1) == 0)
                        {
                                if (temp->data.gender == 'f')
                                {
                                        cout << name1 << "是" << name2 << "的阿姨" << endl;
                                }
                                else
                                {
                                        cout << name1 << "是" << name2 << "的叔叔" << endl;
                                }
                                return 1;
                        }
                        temp = temp->nextsibling;
                }
                silding = silding->nextsibling;
        }
 }
 return 0;
}

void menu(CSTree &T,int &flag)
{
 cout << "
==========================" << endl;
 cout << "                                                                      ||    1.查看家谱                  ||\n";
 cout << "                                                                      ||    2.修改家谱人员信息    ||\n";
 cout << "                                                                      ||    3.查看某人详细信息    ||\n";
 cout << "                                                                      ||    4.将某人开除家谱        ||\n";
 cout << "                                        ||   5.添加家谱成员        ||\n";
 cout << "                                        ||   6.显示第 n 代          ||\n";
 cout << "                                        ||   7.判断两人关系        ||\n";
 cout << "                                        ||   8.按出生查找          ||\n";
 cout << "                                        ||   9.退出                ||" << endl;
 cout << "
==========================" << endl;
 cout << "                                        请输入你的选择: ";
 int choice;
 cin >> choice;
 if (choice == 1)
 {
        TraverseFT(T);
 }
 if (choice == 2)
 {
        modifyinfo(T);
 }
 if (choice == 3)
 {
        Findman(T);
 }
 if (choice == 4)
 {
        Deleteman(T);
 }
 if (choice == 5)
 {
        Insertman(T);
 }
```

```
 if (choice == 6)
 {
        FindNdynasty(T);
 }
 if (choice == 7)
 {
        int t=Show(T);
        if (t == 0)
        {
               cout << "关系超过两代" << endl;
        }
 }
 if (choice == 8)
 {
        Findbirth(T);
 }
 if (choice == 9)
 {
        flag = 0;
 }
}

int main()
{
 CSTree T;
 InitCSTree(T);
 CreateCSTree(T);
 //fclose(stdin);
 int flag = 1;
 while (1)
```

```
{
        menu(T, flag);
        if (flag == 0)
        {
                break;
        }
 }
 free(T);
 free(parent);
 free(silding);
 return 0;
}
```

6. Huffman 编码与解码

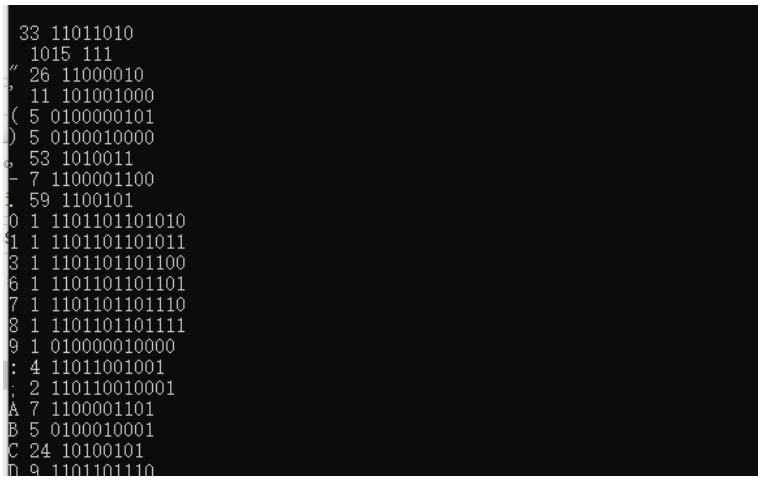
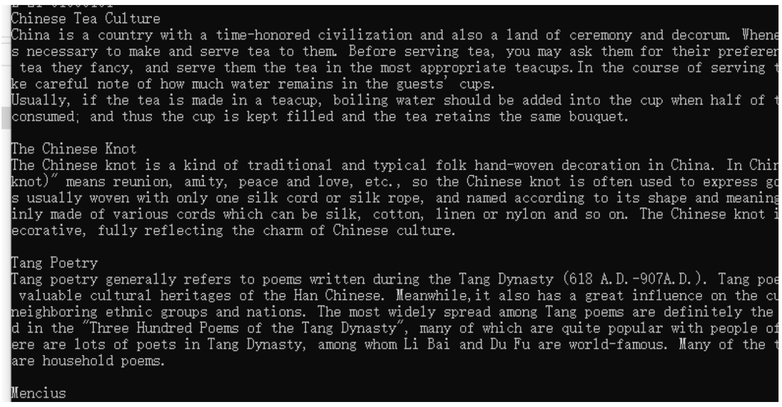
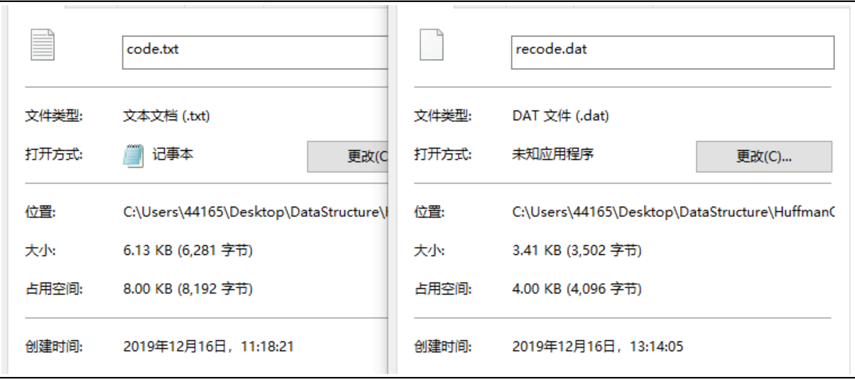
1. 数据结构

   Huffman 树。

2. 设计思想

   Huffman 树的标准操作，在每次找结点的时候遍历找到最小权值的两个节点，然后进行链接。然后根据父子节点关系确定字符的编码，最后写入文件时注意对齐规则，采用满八位写入。

3. 测试数据及样例

编 码	
解 码	
压 缩 对 比	

```
 33 11011010
  1015 111
" 26 11000010
' 11 101001000
( 5 0100000101
) 5 0100010000
, 53 1010011
- 7 1100001100
. 59 1100101
0 1 1101101101010
1 1 1101101101011
3 1 1101101101100
6 1 1101101101101
7 1 1101101101110
8 1 1101101101111
9 1 010000010000
: 4 11011001001
; 2 110110010001
A 7 1100001101
B 5 0100010001
C 24 10100101
D 9 1101101110
```

```
Chinese Tea Culture
China is a country with a time-honored civilization and also a land of ceremony and decorum. Whene
s necessary to make and serve tea to them. Before serving tea, you may ask them for their preferen
 tea they fancy, and serve them the tea in the most appropriate teacups.In the course of serving t
ke careful note of how much water remains in the guests' cups.
Usually, if the tea is made in a teacup, boiling water should be added into the cup when half of t
consumed; and thus the cup is kept filled and the tea retains the same bouquet.

The Chinese Knot
The Chinese knot is a kind of traditional and typical folk hand-woven decoration in China. In Chir
knot)" means reunion, amity, peace and love, etc., so the Chinese knot is often used to express go
s usually woven with only one silk cord or silk rope, and named according to its shape and meaning
inly made of various cords which can be silk, cotton, linen or nylon and so on. The Chinese knot i
ecorative, fully reflecting the charm of Chinese culture.

Tang Poetry
Tang poetry generally refers to poems written during the Tang Dynasty (618 A.D.-907A.D.). Tang poe
 valuable cultural heritages of the Han Chinese. Meanwhile,it also has a great influence on the cu
neighboring ethnic groups and nations. The most widely spread among Tang poems are definitely the
d in the "Three Hundred Poems of the Tang Dynasty", many of which are quite popular with people of
ere are lots of poets in Tang Dynasty, among whom Li Bai and Du Fu are world-famous. Many of the t
are household poems.

Mencius
```

	code.txt	recode.dat
文件类型:	文本文档 (.txt)	DAT 文件 (.dat)
打开方式:	记事本	未知应用程序
位置:	C:\Users\44165\Desktop\DataStructure\	C:\Users\44165\Desktop\DataStructure\HuffmanC
大小:	6.13 KB (6,281 字节)	3.41 KB (3,502 字节)
占用空间:	8.00 KB (8,192 字节)	4.00 KB (4,096 字节)
创建时间:	2019年12月16日，11:18:21	2019年12月16日，13:14:05

4. 算法时间复杂度及优化

时间复杂度为 O(n^2)，如果采用堆排序或优先队列优化为 O(nlogn)。

5. 源代码

```
#include<iostream>
#include<stdlib.h>
#include<stdio.h>
#include<string>
#include<fstream>
```

```cpp
#include<errno.h>
#include<assert.h>
#include<bitset>
using namespace std;
typedef struct HTNode {
  int position;
  int weight;
  int flag;
  string code;
  struct HTNode* parent, * lchild, * rchild;
}HTNode, * HuffmanTree;
int word = 0;
int wordnum=0;
int words[300];
string code[100000];
char article[100000];
void Select(HuffmanTree* HF, int& s1, int& s2, int n)
{
  int k, w,i;
  w = INT_MAX;
  for (k = 1; k <= n; k++)
  {
      if (HF[k]->weight < w && HF[k]->flag == 0 && HF[k]->weight != 0)
      {
          w = HF[k]->weight;
          s1 = k;
      }
  }
  int j = s1;
  w = INT_MAX;
  for (k = 1; k <= n; k++)
  {
      if (HF[k]->weight < w && k != j && HF[k]->flag == 0 && HF[k]->weight != 0)
      {
          w = HF[k]->weight;
          s2 = k;
      }
  }
}

void HuffmanCoding(HuffmanTree* HF, int n)
{
  int s1, s2, i, j;
```

```
    for (i = n + 1; i <= 2 * n; i++)
    {
        Select(HF, s1, s2, i);
        HF[s1]->flag = 1;
        HF[s2]->flag = 1;
        HF[s1]->parent = HF[i];
        HF[s2]->parent = HF[i];
        HF[i]->lchild = HF[s1];
        HF[i]->rchild = HF[s2];
        HF[i]->weight = HF[s1]->weight + HF[s2]->weight;
        HF[i]->flag = 0;
    }
}

void HuffmanCode(HuffmanTree HF)
{
    if (HF->lchild != NULL)
    {
        code[HF->lchild->position] = code[HF->position] + '0';
        HuffmanCode(HF->lchild);
    }
    if (HF->rchild != NULL)
    {
        code[HF->rchild->position] = code[HF->position] + '1';
        HuffmanCode(HF->rchild);
    }
    return;
}

void bianma(HuffmanTree* HF)
{
    fstream file;
    file.open("code.txt", ios::in);
    assert(file.is_open());
    char c;
    int k = 1;
    int t = 0;
    file >> noskipws;
    while (!file.eof())
    {
        file >> c;
        words[c]++;
        article[wordnum] = c;
        wordnum++;
    }
    for (int i = 1; i <= 256; i++)
    {
        if (words[i])
            word++;
    }
    for (int i = 1; i <= 256; i++)
        {
            if (words[i])
            {
                HTNode* p = (HTNode*)malloc(sizeof(struct HTNode));
                p->position = i;
                p->weight = words[i];
                p->lchild = NULL;
                p->rchild = NULL;
                p->flag = 0;
                HF[k] = p;
                k++;
            }
        }
        for (int i = word + 1; i <= 2 * word; i++)
        {
            HTNode* p = (HTNode*)malloc(sizeof(struct HTNode));
            p->position = i+10000;
            p->weight = 0;
            p->lchild = NULL;
            p->rchild = NULL;
            p->flag = 0;
            HF[i] = p;
        }
        HuffmanCoding(HF, word);
        HuffmanCode(HF[2 * word-1]);
        file.close();
    }

    int s = 0;
    char b = 0;
    void WriteBit(char x, FILE* p)//八位存入
    {
        if (x == '1')
            b |= 1 << 7 - s;
        else if (x == '0')
            b |= 0 << 7 - s;
        s++;
        if (s == 8) {
            fwrite(&b, sizeof(char), 1, p);
            b = 0;
            s = 0;
        }
    }

    void WriteCode(HuffmanTree *HF)
    {
        FILE* file;
        errno_t err;
        err=fopen_s(&file,"recode.dat","w");
        for (int i = 0; i < wordnum; i++)
        {
            for (int j = 0; j < code[article[i]].size(); j++)
            {
                WriteBit(code[article[i]][j], file);
            }
        }
        fclose(file);
    }

    void yima(HuffmanTree* HF)
    {
        HTNode* maxn, * temp;
        maxn = (HTNode*)malloc(sizeof(struct HTNode));
        temp = (HTNode*)malloc(sizeof(struct HTNode));
        maxn->weight = 0;
        int i = 0, m = 0;
        maxn = HF[2 * word - 1];
        FILE* fp;
        string rcode;
        errno_t err;
        err = fopen_s(&fp, "recode.dat", "rb");
        char t;
        temp = maxn;
        while (fread(&t, sizeof(char), 1, fp))
        {
            for (int i = 0; i < 8; i++)
            {
                if (((t >> 7) & 1) == 1)
                    rcode += '1';
                else
                    rcode += '0';
                t <<= 1;
            }
        }
        for (int i = 0; i < rcode.size(); i++)
        {
            if (rcode[i] == '0')
            {
                temp = temp->lchild;
                if (temp->lchild == NULL && temp->rchild == NULL)
                {
                    cout << (char)temp->position;
                    temp = maxn;
                }
            }
            else if (rcode[i] == '1')
            {
                temp = temp->rchild;
                if (temp->lchild == NULL && temp->rchild == NULL)
                {
                    cout << (char)temp->position;
                    temp = maxn;
                }
            }
        }
        fclose(fp);
    }

    void shouhuffmancode(HuffmanTree* HF)
    {
        for (int i = 1; i <= word; i++)
        {
            if (HF[i]->weight == 0)
                continue;
            char c = HF[i]->position;
            cout << c << " " << HF[i]->weight << " " << code[HF[i]->position]
 << endl;
        }
    }

    int main()
    {
        int i, n = 256;
        HuffmanTree* HF;
        HTNode* p;
        HF = (HuffmanTree*)malloc((2 * n + 1) * sizeof(HuffmanTree));
        bianma(HF);
        shouhuffmancode(HF);
        WriteCode(HF);
        yima(HF);
    }
```

7. 最小生成树

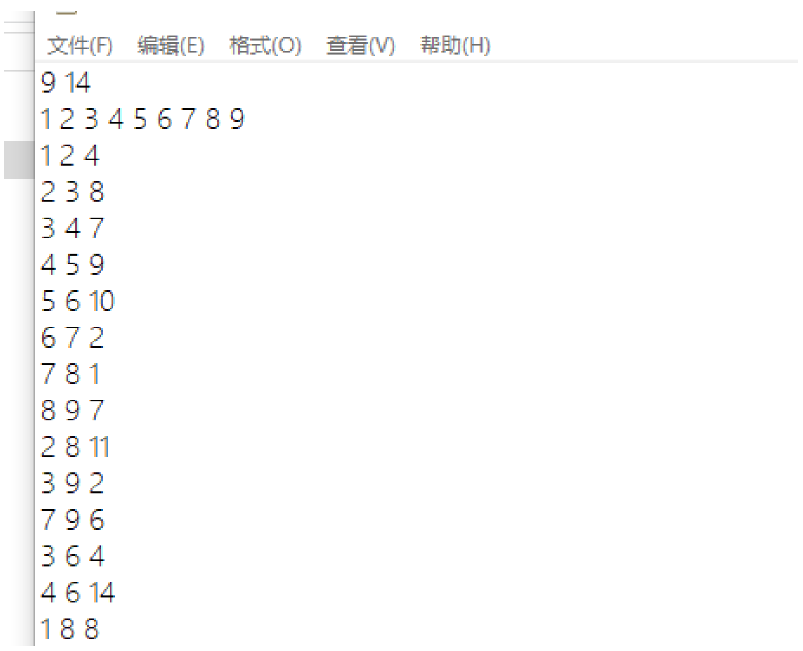
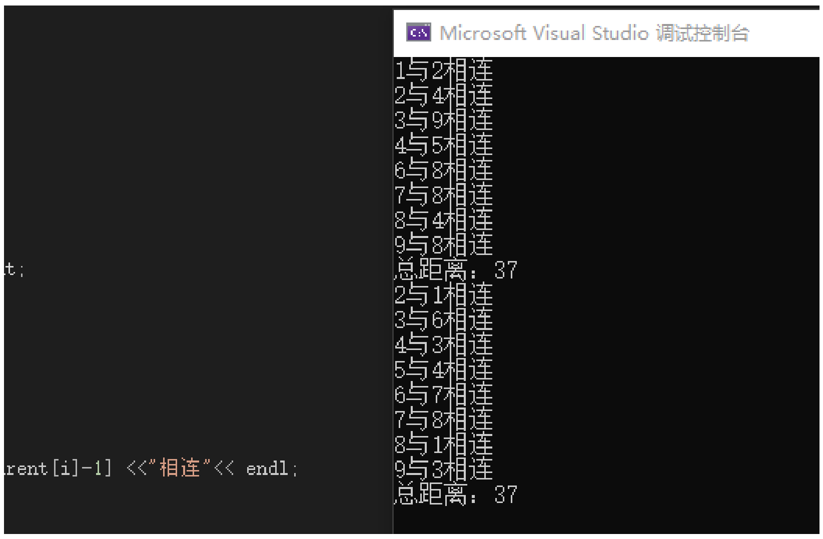
1. 数据结构

   邻接矩阵，并查集。

2. 设计思想

   邻接矩阵的 prim 算法思想为选定起点然后将其余所有节点的当前状况初始化，然后选择权值最小的再对他的相邻未使用过的点点进行权值的更新使得他能获得的权值最小。Kruskal 算法则是运用并查集思想，设置数组存放该点的父亲点，通过 find 函数来确定两条线是否有共同的父亲。并且按照路线的权值排序，从小到大选边判断是否是有相同的父亲，如果不是则相连，及设置一方父亲为另一方父亲，是则继续判断。

3. 样例及测试数据分析

样例 输入	
样例 输出	

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
9 14
1 2 3 4 5 6 7 8 9
1 2 4
2 3 8
3 4 7
4 5 9
5 6 10
6 7 2
7 8 1
8 9 7
2 8 11
3 9 2
7 9 6
3 6 4
4 6 14
1 8 8
```

Microsoft Visual Studio 调试控制台

```
t;
rent[i]-1] <<"相连"<< endl;
```

```
1与2相连
2与4相连
3与9相连
4与5相连
6与8相连
7与8相连
8与4相连
9与8相连
总距离：37
2与1相连
3与6相连
4与3相连
5与4相连
6与7相连
7与8相连
8与1相连
9与3相连
总距离：37
```

4. 算法时间按复杂度及优化

Prim：O(n^2)。

Kruskal：O(mlogm)。

Prim 同 dj 一样可以采用堆优化来使时间复杂度降为 O(nlogn)。

5. 源代码

```
#include<iostream>
#include<stdlib.h>
#include<stdio.h>
#include<algorithm>
#include<vector>
```

```cpp
#include<fstream>
using namespace std;
#define MAX_NUM 20
#define VexType int
#define OK 1
#define ERROR 0
#define QElemType int
typedef enum { DG, DN, UDG, UDN }GraphKind;
struct PrVex {
  float distance;
  VexType vex;
  bool flag;
};
PrVex* Pr = new PrVex[10000000];
typedef struct ArcCell {
  int adj;
  float weight;
}ArcCell, AdjMatrix[MAX_NUM][MAX_NUM];
typedef struct {
  VexType vexs[MAX_NUM];
  AdjMatrix arcs;
  int vexnum, arcnum;
  int kind;
}MGraph;
int Visit[1000001];
typedef struct QNode {
  QElemType data;
  struct QNode* next;
}QNode, * QueuePtr;
typedef struct {
  QueuePtr front;
  QueuePtr rear;
  int length;
}LinkQueue;
vector<char> staname;
int InitQueue(LinkQueue& Q)
{
  Q.front = Q.rear = (QueuePtr)malloc(sizeof(QNode));
  if (!Q.front)
       exit(0);
  Q.front->next = NULL;
  Q.length = 0;
  return OK;
}
```

```

int EnQueue(LinkQueue& Q, int e)
{
  QNode* p = (QueuePtr)malloc(sizeof(QNode));
  if (!p)
      exit(0);
  p->data = e;
  p->next = NULL;
  Q.rear->next = p;
  Q.rear = p;
  Q.length++;
  return OK;
}

int DeQueue(LinkQueue& Q, int& e)
{
  if (Q.front == Q.rear)
      return ERROR;
  QNode* p = Q.front->next;
  e = p->data;
  Q.front->next = p->next;
  if (Q.rear == p)
  {
      Q.rear = Q.front;
  }
  free(p);
  Q.length--;
  return OK;
}

int QueueEmpty(LinkQueue Q)
{
  if (Q.front == Q.rear)
      return 1;
  return 0;
}

int LocateVex(MGraph G, VexType vex)
{
  for (int i = 0; i < G.vexnum; i++)
  {
      if (G.vexs[i] == vex)
          return i;
  }
```

```cpp
return -1;
}

void CreateUDN(MGraph& G,fstream &fp)//wu向图
{
    int vexnum, arcnum;
    VexType start, end;
    int weight;
    fp >> vexnum >> arcnum;
    G.vexnum = vexnum;
    G.arcnum = arcnum;
    for (int i = 0; i < vexnum; i++)
    {
        fp >> G.vexs[i];
    }
    for (int i = 0; i < vexnum; i++)
    {
        for (int j = 0; j < vexnum; j++)
        {
            G.arcs[i][j].adj = INT_MAX;
            G.arcs[i][j].weight = 0;
        }
    }
    for (int i = 0; i < G.arcnum; i++)
    {
        fp >> start >> end >> weight;
        G.arcs[LocateVex(G, start)][LocateVex(G, end)].adj = 0;
        G.arcs[LocateVex(G, start)][LocateVex(G, end)].weight = weight;
        G.arcs[LocateVex(G, end)][LocateVex(G, start)].adj = 0;
        G.arcs[LocateVex(G, end)][LocateVex(G, start)].weight = weight;
    }
}

void CreateMGraph(MGraph& G,fstream &fp)
{
    CreateUDN(G,fp);
}

void Prim(MGraph G, VexType v)
{
    for (int i = 0; i < G.vexnum; i++)
    {
        Pr[i].flag = false;
        Pr[LocateVex(G, v)].flag = true;
        if (G.arcs[LocateVex(G, v)][i].adj == 0)
            Pr[i].distance = G.arcs[LocateVex(G, v)][i].weight;
        else if (i == LocateVex(G, v))
            Pr[i].distance = 0;
        else
            Pr[i].distance = INT_MAX;
        Pr[i].vex = v;
    }
    while (1)
    {
        int sum = 0;
        int minnposition;
        float minn = INT_MAX;
        for (int i = 0; i < G.vexnum; i++)
        {
            if (Pr[i].flag == true)
                continue;
            if (minn >= Pr[i].distance)
            {
                minnposition = i;
                minn = Pr[i].distance;
            }
        }
        Pr[minnposition].flag = true;
        v = G.vexs[minnposition];//找到最短的distance
        for (int i = 0; i < G.vexnum; i++)
        {
            if (Pr[i].flag == true)//已经遍历的点不在遍历
                continue;
            if (G.arcs[LocateVex(G, v)][i].adj == 0)
            {
                if (G.arcs[LocateVex(G, v)][i].weight < Pr[i].distance)
                {
                    Pr[i].vex = v;
                    Pr[i].distance = G.arcs[LocateVex(G, v)][i].weight;
                }
            }
        }
        for (int i = 0; i < G.vexnum; i++)
        {
            if (Pr[i].flag == true)
                sum++;
        }
        if (sum == G.vexnum)
```

```
                break;
        }
}

void showPrim(MGraph G)
{
    float sum = 0;
    for (int i = 0; i < G.vexnum; i++)
    {
        sum += Pr[i].distance;
        if (Pr[i].distance == 0)
            continue;
        else
            cout << G.vexs[i] << "与" << Pr[i].vex << "相连\n";
    }
    cout << "总距离：" << sum << endl;
}

struct Edge {
    int start;
    int end;
    float weight;
}E[100001];
int N, M;
int parent[10001];
bool cmp(Edge a, Edge b)
{
    return a.weight < b.weight;
}

int finda(int x)
{
    if (parent[x] == x)
        return x;
    else
        return finda(parent[x]);
}

float kruskal()
{
    float ans = 0;
    int t = 0;
    for (int i = 1; i <= M; i++)
    {
```

```cpp
        int t1 = finda(E[i].start);
        int t2 = finda(E[i].end);
        if (t1 == t2)
            continue;
        else if (t1 != t2)
        {
            t++;
            ans += E[i].weight;
            parent[t1] = parent[t2];
        }
    }
    if (t < N - 1)
        return 0;
    return ans;
}

int main()
{
    fstream fp;
    fp.open("3.txt", ios::in);
    char name;
    fp >> N >> M;
    for (int i = 1; i <= N; i++)
    {
        parent[i] = i;
    }
    for (int i = 1; i <= N; i++)
    {
        fp >> name;
        staname.push_back(name);
    }
    for (int i = 1; i <= M; i++)
    {
        fp >> E[i].start >> E[i].end >> E[i].weight;
    }
    sort(E + 1, E + M + 1, cmp);
    float ans=kruskal();
    for (int i = 1; i <= N; i++)
    {
        if (i == parent[i])
            continue;
        cout << staname[i-1] << "与" << staname[parent[i]-1] <<"相连"<< endl;
    }
    cout << "总距离：" << ans << endl;
    fp.close();
    fp.open("3.txt", ios::in);
    MGraph G;
    CreateMGraph(G, fp);
    Prim(G, 1);
    showPrim(G);
    fp.close();
}
```

8. 公交线路提示

1. 数据结构

   邻接表和优先队列。

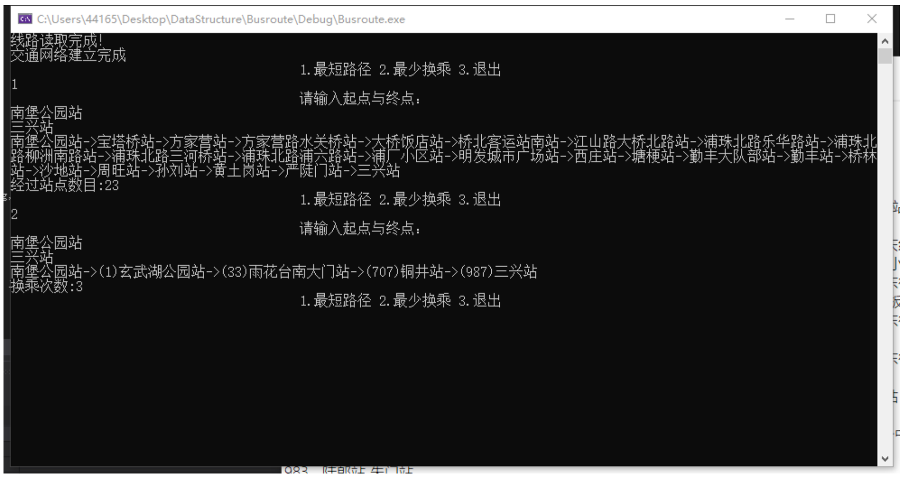
2. 设计思想

   ① 最短距离：普通的迪杰斯特拉算法，但是站点过多会爆慢，选择使用堆优化及优先队列，将 dj 结构体按照距离权值出队。

   ② 最少换乘：同样使用迪杰斯特拉算法，初始化在同一公交线路上的站点的最小距离要么被更新过，要么与其他保持相同，与起始点在同一公交线路上的均为 0，换一路则+1。

   ③ 站点实行编号，通过编号直接调用站点，而不是通过站点调用下标，再使用，这样减少了查找站点位置的时间，同时也加快了文件的读写速度，并且用 map 进行存储去重。

3. 样例及测试数据分析



```
C:\Users\44165\Desktop\DataStructure\Busroute\Debug\Busroute.exe
线路读取完成!
交通网络建立完成
                                        1.最短路径 2.最少换乘 3.退出

1
                                        请输入起点与终点:

南堡公园站
三兴站
南堡公园站->宝塔桥站->方家营站->方家营路水关桥站->大桥饭店站->桥北客运站南站->江山路大桥北路站->浦珠北路乐华路站->浦珠北
路柳洲南路站->浦珠北路三河桥站->浦珠北路浦六路站->浦厂小区站->明发城市广场站->西庄站->塘梗站->勤丰大队部站->勤丰站->桥林
站->沙地站->周旺站->孙刘站->黄土岗站->严陡门站->三兴站
经过站点数目:23
                                        1.最短路径 2.最少换乘 3.退出

2
                                        请输入起点与终点:

南堡公园站
三兴站
南堡公园站->(1)玄武湖公园站->(33)雨花台南大门站->(707)铜井站->(987)三兴站
换乘次数:3
                                        1.最短路径 2.最少换乘 3.退出
```

选取南堡公园站为起点，三兴站作为终点。

4. 算法时间复杂度及优化

迪杰斯特拉经过堆优化后时间复杂度为 O(nlogn)，在优化过后，相较于之前三秒出结果甚至十一秒出结果要快非常多，所有答案都是秒出。

5. 源代码

```cpp
#include<iostream>
#include<map>
#include<string>
#include<stdlib.h>
#include<stdio.h>
#include<vector>
#include<map>
#include<queue>
#include<fstream>
#include<ctime>
#include<set>
using namespace std;
struct DjNode {
  int distance;
  int pre;
  bool flag;
  int name;
  int prebus;
  DjNode() {
        distance = INT_MAX-1;
        flag = false;
    }
    const bool operator < (const DjNode& a) const {
        return a.distance < distance;
    }
};
map<int,string> mp;
map<string,int> mp2;
map<int, int> buss;
vector<int> bus[1000];
int *busname=(int*)malloc(sizeof(int) * 1000);
int busnum=1;
int stationnum = 0;
int InitBus()
{
    fstream fp;
    fp.open("南京公交线路.txt", ios::in);
    int i=0;
    int k = 0;
    string temp;
    fp >> busname[i];
    buss.insert({ busname[i],i });
    while (!fp.eof())
    {
        while (!fp.eof())
        {
            fp >> temp;
            int flag = 0;
            for (int j = 0; j < temp.length(); j++)
            {
                int flag1 = 0;
                if (temp[j]<’0’ || temp[j]>’9’)
                {
                    flag = 1;
                    map<string, int>::iterator it;
                    it = mp2.find(temp);
                    if (it!=mp2.end())
                    {
                        bus[i].push_back(it->second);
                        flag1 = 1;
                        break;
                    }
                    if (flag1 == 1)
                        break;
                    bus[i].push_back(stationnum);
                    mp.insert({ stationnum,temp });
                    mp2.insert({ temp,stationnum });
                    stationnum++;
                    break;
                }
            }
            if (flag == 0)
            {
                i++;
                busnum++;
                busname[i] = stoi(temp);
                buss.insert({ busname[i],i });
                break;
            }
        }
    }
    cout << "线路读取完成！" << endl;
    fp.close();
    return 1;
}

DjNode* Dj = new DjNode[6000];
typedef struct {
    int name;
    vector<vector<int>> bus;
    vector<int> vicinity;
}MNode;
typedef struct {
    vector<MNode> Graph;
    int vexnum;
}MGraph;

void CreateGraph(MGraph& G)
{
    int start, end;
    auto it = mp.begin();
    vector<vector<int>> nothing;
    vector<int> nope;
    for (int i = 0; i < mp.size(); i++)
    {
        G.Graph.push_back({ it->first,nothing,nope });
        it++;
    }
    G.vexnum = mp.size();
    for (int i = 0; i < busnum; i++)
    {
        for (int j = 0; j < bus[i].size()-1; j++)
        {
            int flag = 0,flag2=0;
            start = bus[i][j];
            end = bus[i][j + 1];
            for (int t = 0; t < G.Graph[start].vicinity.size(); t++)
            {
                if (G.Graph[start].vicinity[t] == end)
                {
                    G.Graph[start].bus[t].push_back(busname[i]);
                    flag = 1;
                }
            }
            for (int k = 0; k < G.Graph[end].vicinity.size(); k++)
            {
                if (G.Graph[end].vicinity[k] == start)
                {
                    G.Graph[end].bus[k].push_back(busname[i]);
                    flag = 1;
                }
            }
            if (flag == 0)
            {
                G.Graph[start].vicinity.push_back(end);
                G.Graph[start].bus.push_back({ busname[i] });
            }
            if (flag2 == 0)
            {
                G.Graph[end].vicinity.push_back(start);
                G.Graph[end].bus.push_back({ busname[i] });
            }
        }
    }
    cout << "交通网络建立完成" << endl;
}
priority_queue <DjNode> q;
void Dijkstra(MGraph G, int start)
{
    for (int i = 0; i < G.vexnum; i++)
    {
        Dj[i].flag = false;
        Dj[i].name = G.Graph[i].name;
        Dj[i].distance = INT_MAX - 1;
        Dj[i].pre = start;
    }
    Dj[start].distance = 0;
    Dj[start].pre = start;
    q.push(Dj[start]);
    while (!q.empty())
    {
        DjNode temp = q.top();
        q.pop();
        if (Dj[temp.name].flag == true)
            continue;
        Dj[temp.name].flag = true;
        for (int i = 0; i < G.Graph[temp.name].vicinity.size(); i++)
        {
            if (Dj[G.Graph[temp.name].vicinity[i]].flag == true)
                continue;
            if (Dj[G.Graph[temp.name].vicinity[i]].distance > Dj[temp.name].distance + 1)
            {
                Dj[G.Graph[temp.name].vicinity[i]].distance = Dj[temp.name].distance + 1;
                Dj[G.Graph[temp.name].vicinity[i]].pre = Dj[temp.name].name;
                q.push(Dj[G.Graph[temp.name].vicinity[i]]);
            }
        }
    }
}

void leastchange(MGraph G, int start)
{
    int level = 0;
    for (int i = 0; i < G.vexnum; i++)
    {
        Dj[i].flag = false;
        Dj[i].name = G.Graph[i].name;
        Dj[i].distance = INT_MAX - 1;
        Dj[i].pre = start;
        Dj[i].prebus = 0;
    }
    Dj[start].distance = 0;
    Dj[start].pre = start;
    q.push(Dj[start]);
    while (!q.empty())
    {
        DjNode temp = q.top();
        q.pop();
        if (Dj[temp.name].flag == true)
            continue;
        Dj[temp.name].flag = true;
        for (int i = 0; i < G.Graph[temp.name].bus.size(); i++)
        {
            for (int j = 0; j < G.Graph[temp.name].bus[i].size(); j++)
            {
                int tempbus = G.Graph[temp.name].bus[i][j];
                int pos = buss[tempbus];
                for (int k = 0; k < bus[pos].size(); k++)
                {
                    if (bus[pos][k] == temp.name)
                    {
                        int s;
                        if (level != 0)
                        {
                            for (s = 0; s < bus[pos].size(); s++)
                            {
                                if (Dj[bus[pos][s]].flag == true)
                                    continue;
                                if (Dj[temp.name].distance+1 < Dj[bus[pos][s]].distance)
                                {
                                    Dj[bus[pos][s]].distance = Dj[temp.name].distance+1;
                                    Dj[bus[pos][s]].pre = temp.name;
                                    q.push(Dj[bus[pos][s]]);
                                    Dj[bus[pos][s]].prebus = tempbus;
                                }
                            }
                        }
                        else if (level == 0)
                        {
                            for (s = 0; s < bus[pos].size(); s++)
                            {
                                if (Dj[bus[pos][s]].flag == true)
                                    continue;
                                Dj[bus[pos][s]].distance = 0;
                                Dj[bus[pos][s]].pre = temp.name;
                                q.push(Dj[bus[pos][s]]);
                                Dj[bus[pos][s]].prebus = tempbus;
                            }
                        }
                        if (s == bus[pos].size())
                            break;
                    }
                }
            }
        }
        level++;
    }
}

int SearchsPath(MGraph G, int start, int end)
{
    int v = end;
    int *a=new int[5000];
    int sum = 0;
    if (Dj[end].distance == INT_MAX-1)
    {
        cout << "Cant Reach!" << endl;
        return 0;
    }
    while (v != start)
    {
        a[sum] = v;
        v = Dj[v].pre;
        sum++;
    }
    cout << mp[start] << "->";
    for (int i = sum - 1; i >= 1; i--)
    {
        cout << mp[a[i]] << "->";
    }
    cout << mp[a[0]] << endl;
    cout << "经过站点数目:" << Dj[end].distance << endl;
    return 1;
}

int SearchlPath(MGraph G, int start, int end)
{
    int v = end;
    int* a = new int[5000];
```

```cpp
    int sum = 0;
    if (Dj[end].distance == INT_MAX - 1)
    {
        cout << "Cant Reach!" << endl;
        return 0;
    }
    while (v != start)
    {
        a[sum] = v;
        v = Dj[v].pre;
        sum++;
    }
    cout << mp[start] << "->";
    for (int i = sum - 1; i >= 1; i--)
    {
        cout <<"("<<Dj[a[i]].prebus<<")" <<mp[a[i]] << "->";
    }
    cout << "(" << Dj[a[0]].prebus << ")" <<mp[a[0]] << endl;
    cout << "换乘次数:" << Dj[end].distance << endl;
    return 1;
}

int main()
{
    MGraph G;
    InitBus();
    CreateGraph(G);
    string start,end;
    int a, b;
    while (1)
    {
        cout << "                                        1.最短路径" << "
" << "2.最少换乘" <<" 3.退出"<< endl;
        int choice;
        cin >> choice;
        if (choice == 1)
        {
            cout << "                                        请输入起点
与终点：" << endl;
            cin >> start >> end;
            Dijkstra(G, mp2[start]);
            SearchsPath(G, mp2[start], mp2[end]);
        }
        else if (choice == 2)
```

```
            {
                cout << "请输入起点与终点：" << endl;
                cin >> start >> end;
                leastchange(G, mp2[start]);
                Search1Path(G, mp2[start], mp2[end]);
            }
            else
                break;
        }
        return 0;
}
```

9. 排序算法比较

1. 数据结构

线性表（顺序-普通排序，链式-基数排序）。

2. 设计思想

基数排序是每次通过计算当前数的当前所需位数然后按当前位数存入链表数组中，存完后通过一条线性链表直接连结起来，然后继续比对。

快速排序则通过将比第一个数小的数移到他左边，大的在右，重复到区间只有一个元素。

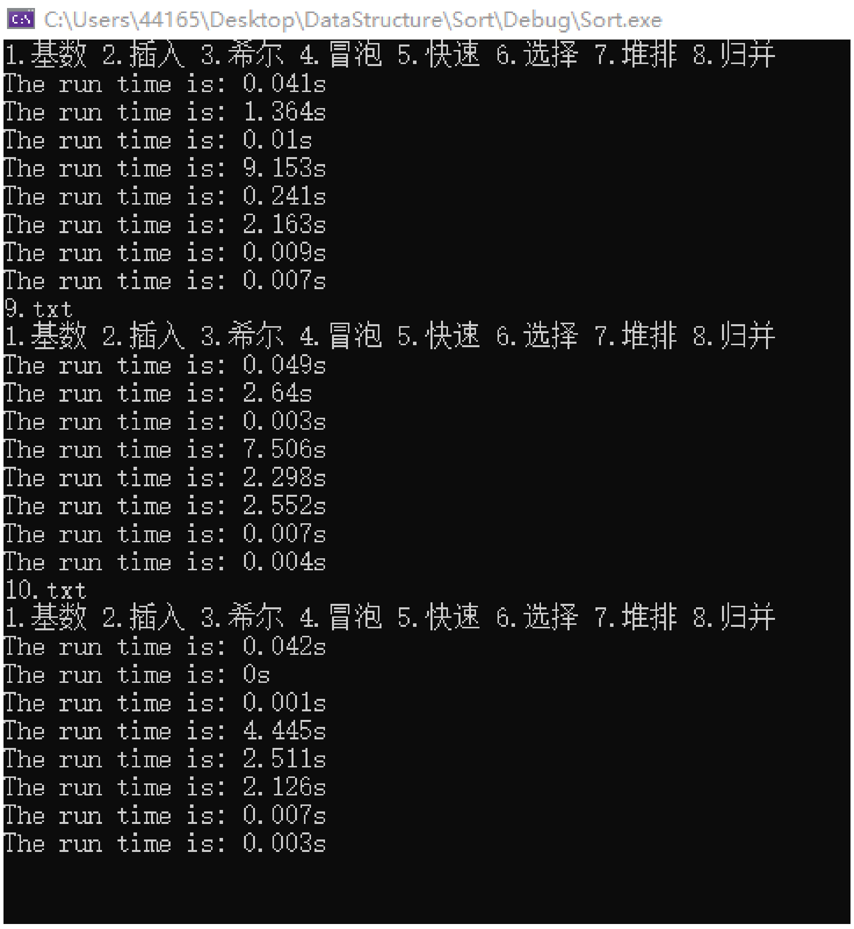
归并排序则是采用两个线性表，每次将两个相邻区间的数比较按照大小存入另一个，直到相邻区间采用间隔过大。

堆排序则是将第一个数移到末尾并对数组再次大顶堆排序，重复至最后一个数，得到的结果为从小到大排序。

其余的都为简单排序。

然后调用排序时计算运行时间。

3. 样例及测试数据分析

1.txt-乱序 9.txt-倒序 10.txt-顺序	

```
C:\Users\44165\Desktop\DataStructure\Sort\Debug\Sort.exe
1.基数 2.插入 3.希尔 4.冒泡 5.快速 6.选择 7.堆排 8.归并
The run time is: 0.041s
The run time is: 1.364s
The run time is: 0.01s
The run time is: 9.153s
The run time is: 0.241s
The run time is: 2.163s
The run time is: 0.009s
The run time is: 0.007s
9.txt
1.基数 2.插入 3.希尔 4.冒泡 5.快速 6.选择 7.堆排 8.归并
The run time is: 0.049s
The run time is: 2.64s
The run time is: 0.003s
The run time is: 7.506s
The run time is: 2.298s
The run time is: 2.552s
The run time is: 0.007s
The run time is: 0.004s
10.txt
1.基数 2.插入 3.希尔 4.冒泡 5.快速 6.选择 7.堆排 8.归并
The run time is: 0.042s
The run time is: 0s
The run time is: 0.001s
The run time is: 4.445s
The run time is: 2.511s
The run time is: 2.126s
The run time is: 0.007s
The run time is: 0.003s
```

4. 算法时间复杂度及优化

时间复杂度：直接插入排序：O(n^2)。希尔插入排序：O(n^1.3)。直接选择排序：O(n^2)。堆排序：O(nlogn)。冒泡排序：O(n^2)。快速排序：O(nlogn)。归并排序：O(nlogn)。基数排序：O(d(r+n))。其中都有最好和最坏情况。如插入排序最好情况为顺序，最坏为倒序。

优化：为防止爆栈，可通过堆栈和队列将递归函数转化为非递归函数，防止爆栈问题。

5. 源代码

```
#include<iostream>
#include<stdlib.h>
#include<stdio.h>
```

```
#include<fstream>
#include<queue>
#include<ctime>
#define Listsize 50001
#define increment 10
#define RADIX 10
#define MAX_SPACE 10000
using namespace std;
typedef struct LNode {
  int* elem;
  int listsize;
  int length;
}List;
typedef struct lnode {
  int data;
  lnode* next;
}lnode,*list;
int Initlist(list& l)
{
  l = (lnode*)malloc(sizeof(struct lnode));
  if (!l)
        return 0;
  l->data = 0;
  l->next = NULL;
  return 1;
}

int insert(list& l, int data)
{
  lnode *temp = l;
```

```
    lnode* p = (lnode*)malloc(sizeof(struct lnode));
    while (temp->next)
    {
        temp = temp->next;
    }
    p->data = data;
    p->next = NULL;
    temp->next = p;
    l->data++;
    return 1;
}

int Delete(list& l, int& data)
{
    if (l->next == NULL)
        return 0;
    lnode* temp = l->next;
    l->next = temp->next;
    data = temp->data;
    l->data--;
    return 1;
}

int freelist(list& l)
{
    int i;
    while (l->data != 0)
        Delete(l, i);
    return 1;
}
```

```
list e[10], f[10];
int Keynum(int data, int n)
{
  int j = 1;
  while (n--)
  {
        j *= 10;
  }
  data = data % j;
  data = data / (j / 10);
  return data;
}

int Set(list& e, list& f, int data)
{
  lnode* p = (lnode*)malloc(sizeof(struct lnode));
  lnode* temp;
  p->data = data;
  p->next = NULL;
  if (f->next == NULL)
  {
        f->next = p;
        e->next = p;
        f->data++;
        e->data++;
        return 1;
  }
  else if (f->next != NULL)
  {
        e->next->next = p;
```

```
            e->next = p;
            f->data++;
            e->data++;
    }
    return 1;
}
clock_t a, b;
time_t temp = 0;
int Select(list& l, int k)
{
    int data;
    Delete(l, data);
    a = clock();
    int key = Keynum(data, k);
    Set(e[key], f[key], data);
    b = clock();
    temp += b - a;
    return 1;
}

int rel(list& l)
{
    lnode *temp = l;
    for (int i = 0; i <= 9; i++)
    {
            temp->next = f[i]->next;
            f[i]->next = NULL;
            e[i]->next = NULL;
            while (temp->next)
            {
```

```
            temp = temp->next;
        }
    }
    return 1;
}

void Traver(list l)
{
    lnode* temp = l->next;
    while (temp)
    {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

void RadixSort(list& l,int dk)
{
    int length = l->data;
    for (int i = 1; i <= dk; i++)
    {
        for (int j = 0; j <= 9; j++)
        {
            f[j]->data = 0;
            e[j]->data = 0;
        }
        while (l->data != 0)
        {
            Select(l, i);
            }
            rel(l);
            l->data = length;
    }
}
//以上均为基数排序

int InitList(List& L)
{
    L.elem = (int*)malloc(sizeof(int)*Listsize);
    if (!L.elem)
            exit(0);
    L.listsize = Listsize;
    L.length = 0;
    return 1;
}

void freelist(List& L)
{
    delete L.elem;
    L.elem = NULL;
    L.length = 0;
}

int Insert(List& L,int e,int i)
{
    if (L.elem == NULL)
            cout << "No exist the list!" << endl;
    else if (i<1 || i>L.length + 1) {
            cout << "Cannot be inserted!" << endl;
    }
    else {
            if (L.length >= L.listsize)
            {
                    int* newbase = (int*)realloc(L.elem, (L.listsize + increment) * sizeof(int));
                    if (!newbase)
                            exit(0);
                    L.elem = newbase;
                    L.listsize += increment;
            }
            int j;
            for (j = L.length; j > i; j--)
            {
                    L.elem[j + 1] = L.elem[j];
            }
            L.elem[i - 1] = e;
            L.length++;
    }
    return 1;
}

void Traverse(List L)
{
    for (int i = 0; i < L.length; i++)
            cout << L.elem[i] << " ";
    cout << endl;
}

void InsertSort(List& L)
{
  int temp,j;
  for (int i = 1; i < L.length; i++)
  {
        if (L.elem[i] < L.elem[i - 1])
        {
              temp = L.elem[i];
              L.elem[i] = L.elem[i - 1];
              for (j = i - 2; temp < L.elem[j]; j--)
              {
                    L.elem[j + 1] = L.elem[j];
              }
              L.elem[j + 1] = temp;
        }
  }
}

void ShellInsert(List& L,int dk)
{
  int temp, j;
  for (int i = dk; i < L.length; i++)
  {
        if (L.elem[i] < L.elem[i - dk])
        {
              temp = L.elem[i];
              L.elem[i] = L.elem[i - dk];
              for (j = i - dk; j >= 0 && (temp < L.elem[j]); j -= dk)
                    L.elem[j + dk] = L.elem[j];
              L.elem[j + dk] = temp;
        }
  }
```

```
}

void ShellSort(List& L,int n)
{
  int num = n,t=0,dlta=1,temp;
  while (num != 0)
  {
        t++;
        num /= 2;
  }
  temp = t + 1;
  t--;
  while (t)
  {
        dlta *= 2;
        t--;
  }
  dlta *= 2;
  for (int k = 0; k < temp; k++)
  {
        dlta /= 2;
        ShellInsert(L, dlta + 1);
  }
}

void BubbleSort(List& L)
{
  for (int i = 0; i < L.length; i++)
  {
        for (int j = 0; j < L.length-1; j++)
		{
			if (L.elem[j] > L.elem[j + 1])
			{
				int temp = L.elem[j];
				L.elem[j] = L.elem[j + 1];
				L.elem[j + 1] = temp;
			}
		}
	}
}

int PartSort(List& L, int low, int high)
{
	int temp;
	temp = L.elem[low];
	int i = low, j = high;
	while (i < j)
	{
		while (i < j && L.elem[j] >= temp)
			j--;
		L.elem[i] = L.elem[j];
		while (i < j && L.elem[i] <= temp)
			i++;
		L.elem[j] = L.elem[i];
	}
	L.elem[i] = temp;
	return i;
}

void QuickSort(List& L,int low,int high)
```

```
{
  queue<int> part;
  int mid = PartSort(L, 0, L.length - 1);
  part.push(0);
  part.push(mid-1);
  part.push(mid + 1);
  part.push(L.length - 1);
  while (!part.empty())
  {
        int start = part.front();
        part.pop();
        int end = part.front();
        part.pop();
        if (start == end||end<start)
              continue;
        mid = PartSort(L, start, end);
        part.push(start);
        part.push(mid - 1);
        part.push(mid + 1);
        part.push(end);
  }
}

void SelectSort(List& L)
{
  int minposition, minn;
  for (int i = 0; i < L.length; i++)
  {
        minn = L.elem[i];
        minposition = i;
```

```
            for (int j = i; j < L.length; j++)
            {
                if (L.elem[j] < minn)
                {
                    minn = L.elem[j];
                    minposition = j;
                }
            }
            if (minposition != i)
            {
                int temp = L.elem[i];
                L.elem[i] = L.elem[minposition];
                L.elem[minposition] = temp;
            }
    }
}

void HeapAdjust(List& L, int s, int m)
{
    int temp = L.elem[s];
    for (int j = 2 * s+1; j <= m; j =2*j+1)
    {
            if (j < m && L.elem[j] < L.elem[j + 1])
                j++;
            if (temp > L.elem[j])
                break;
            L.elem[s] = L.elem[j];
            s = j;
    }
    L.elem[s] = temp;
```

```
}

void HeapSort(List& L)
{
   for (int i = (L.length - 1) / 2; i >= 0; i--)
           HeapAdjust(L, i, L.length - 1);
   for (int i = L.length - 1; i >= 1; i--)
   {
           int temp = L.elem[0];
           L.elem[0] = L.elem[i];
           L.elem[i] = temp;
           HeapAdjust(L, 0, i - 1);
   }
}
int level = 0;
void Merge(List L, List& L1, int i, int m, int n)
{
   int k = i, j = m + 1, t = i;
   if (i == L.length - 1)
   {
           L1.elem[i] = L.elem[i];
   }
   else
   {
           while (t <= m && j <= n)
           {
                   if (L.elem[t] < L.elem[j])
                   {
                           L1.elem[k] = L.elem[t];
                           t++;
```

```
              k++;
          }
          else
          {
              L1.elem[k] = L.elem[j];
              k++;
              j++;
          }
      }
      while (t <= m)
      {
          L1.elem[k] = L.elem[t];
          t++;
          k++;
      }
      while (j <= n)
      {
          L1.elem[k] = L.elem[j];
          j++;
          k++;
      }
  }
}

void MSort(List& L, List& L1)
{
  int n = 1;
  level = 0;
  while (n < L.length)
  {
      int t = 0, s = 0, m = 0;
      while (t < L.length)
      {
          m = t + n - 1;
          s = m + n;
          if (m > L.length - 1)
          {
              m = L.length - 2;
          }
          if (s > L.length - 1)
          {
              s = L.length - 1;
          }
          if (s - L.length + 1 <= n && s - L.length + 1 > 0)
          {
              s = L.length - 1;
          }
          if (level % 2 == 0)
          {
              Merge(L, L1, t, m, s);
          }
          else
          {
              Merge(L1, L, t, m, s);
          }
          t = s + 1;
      }
      n *= 2;
      level++;
  }
```

```
}

void RadixUse(fstream &fp)
{
  int max = -10000;
  list l;
  Initlist(l);
  for (int i = 0; i <= 9; i++)
  {
        Initlist(f[i]);
        Initlist(e[i]);
  }
  int n, data;
  fp >> n;
  for (int i = 1; i <= n; i++)
  {
        fp >> data;
        if (data > max)
              max = data;
        insert(l, data);
  }
  int j = 0;
  while (max)
  {
        j++;
        max /= 10;
  }
  clock_t startTime, endTime;
  startTime = clock();//计时开始
  RadixSort(l, j);
```

```
    endTime = clock();
    cout << "The run time is: " << (double)(endTime - startTime - temp) /
CLOCKS_PER_SEC << "s" << endl;
    temp = 0;
    //Traver(l);
    delete l;
  }
  List L, L1;
  void InsertUse(fstream& fp)
  {
    int n,data;
    fp >> n;
    InitList(L);
    for (int i = 1; i <= n; i++)
    {
        fp >> data;
        Insert(L, data, i);
    }
    clock_t startTime, endTime;
    startTime = clock();//计时开始
    InsertSort(L);
    endTime = clock();
    cout << "The run time is: " << (double)(endTime - startTime) / CLOCKS_PER_SEC
<< "s" << endl;
    //Traverse(L);
    freelist(L);
  }

  void ShellUse(fstream& fp)
  {
    int n, data;
    fp >> n;
    InitList(L);
    for (int i = 1; i <= n; i++)
    {
        fp >> data;
        Insert(L, data, i);
    }
    clock_t startTime, endTime;
    startTime = clock();//计时开始
    ShellSort(L,n);
    endTime = clock();
    cout << "The run time is: " << (double)(endTime - startTime) / CLOCKS_PER_SEC
<< "s" << endl;
    //freelist(L);
    freelist(L);
  }

  void BubbleUse(fstream& fp)
  {
    int n, data;
    fp >> n;
    InitList(L);
    for (int i = 1; i <= n; i++)
    {
        fp >> data;
        Insert(L, data, i);
    }
    clock_t startTime, endTime;
    startTime = clock();//计时开始
```

```
	BubbleSort(L);
	endTime = clock();
	cout << "The run time is: " << (double)(endTime - startTime) / CLOCKS_PER_SEC
<< "s" << endl;
	//Traverse(L);
	freelist(L);
}

void QuickUse(fstream& fp)
{
	int n, data;
	fp >> n;
	InitList(L);
	for (int i = 1; i <= n; i++)
	{
		fp >> data;
		Insert(L, data, i);
	}
	clock_t startTime, endTime;
	startTime = clock();//计时开始
	QuickSort(L,0,n-1);
	endTime = clock();
	cout << "The run time is: " << (double)(endTime - startTime) / CLOCKS_PER_SEC
<< "s" << endl;
	//Traverse(L);
	freelist(L);
}

void SelectUse(fstream& fp)
{
    int n, data;
    fp >> n;
    InitList(L);
    for (int i = 1; i <= n; i++)
    {
        fp >> data;
        Insert(L, data, i);
    }
    clock_t startTime, endTime;
    startTime = clock();//计时开始
    SelectSort(L);
    endTime = clock();
    cout << "The run time is: " << (double)(endTime - startTime) / CLOCKS_PER_SEC
<< "s" << endl;
    //Traverse(L);
    freelist(L);
  }

  void HeapUse(fstream& fp)
  {
    int n, data;
    fp >> n;
    InitList(L);
    for (int i = 1; i <= n; i++)
    {
        fp >> data;
        Insert(L, data, i);
    }
    clock_t startTime, endTime;
    startTime = clock();//计时开始
    HeapSort(L);
    endTime = clock();
    cout << "The run time is: " << (double)(endTime - startTime) / CLOCKS_PER_SEC
<< "s" << endl;
    //Traverse(L);
    freelist(L);
  }

  void MergeUse(fstream& fp)
  {
    InitList(L);
    InitList(L1);
    int n, data;
    fp >> n;
    L1.length = n;
    for (int i = 1; i <= n; i++)
    {
        fp >> data;
        Insert(L, data, i);
    }
    clock_t startTime, endTime;
    startTime = clock();//计时开始
    MSort(L,L1);
    endTime = clock();
    cout << "The run time is: " << (double)(endTime - startTime) / CLOCKS_PER_SEC
<< "s" << endl;
    /*if (level % 2 == 0)
        Traverse(L);
    else
        Traverse(L1);*/
    freelist(L);

    freelist(L1);

  }


  int main()

  {

    fstream fp;

    while (1)

    {

          string file;

          cin >> file;

          if (file == "#")

          {

                break;

          }

          cout << "1.基数  2.插入  3.希尔  4.冒泡  5.快速  6.选择  7.堆排  8.归并" <<
endl;

          fp.open(file, ios::in);

          RadixUse(fp);

          fp.close();

          fp.open(file, ios::in);

          InsertUse(fp);

          fp.close();

          fp.open(file, ios::in);

          ShellUse(fp);

          fp.close();

          fp.open(file, ios::in);

          BubbleUse(fp);

          fp.close();

          fp.open(file, ios::in);
```

```
            QuickUse(fp);
            fp.close();
            fp.open(file, ios::in);
            SelectUse(fp);
            fp.close();
            fp.open(file, ios::in);
            HeapUse(fp);
            fp.close();
            fp.open(file, ios::in);
            MergeUse(fp);
            fp.close();
        }
    }
```

10. 朋友圈

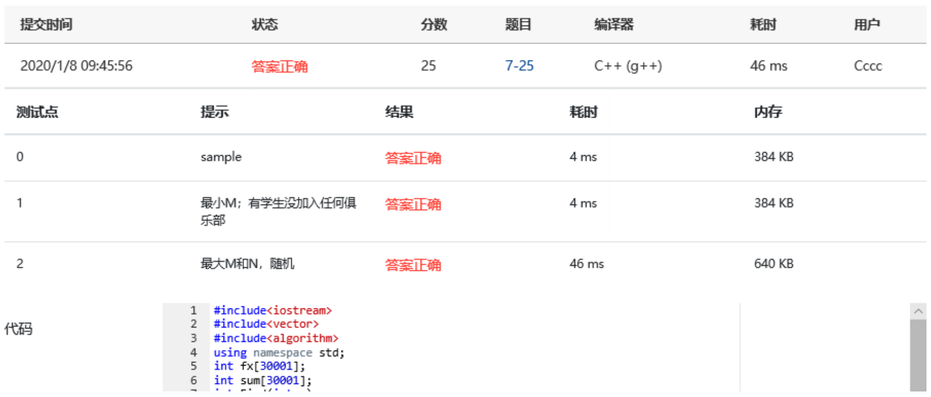
1. 数据结构

   并查集

2. 设计思想

   运用并查集，在同一社团的人，将所有人的父亲设置为第一个人，然后通过寻找他们最深处的父亲，及父亲的父亲的父亲的..，直到此人的父亲是自己。然后在该人的编号为下标的数组处进行自加，最后找到人数最多的，即是最多认识的人。

3. 样例及测试数据分析

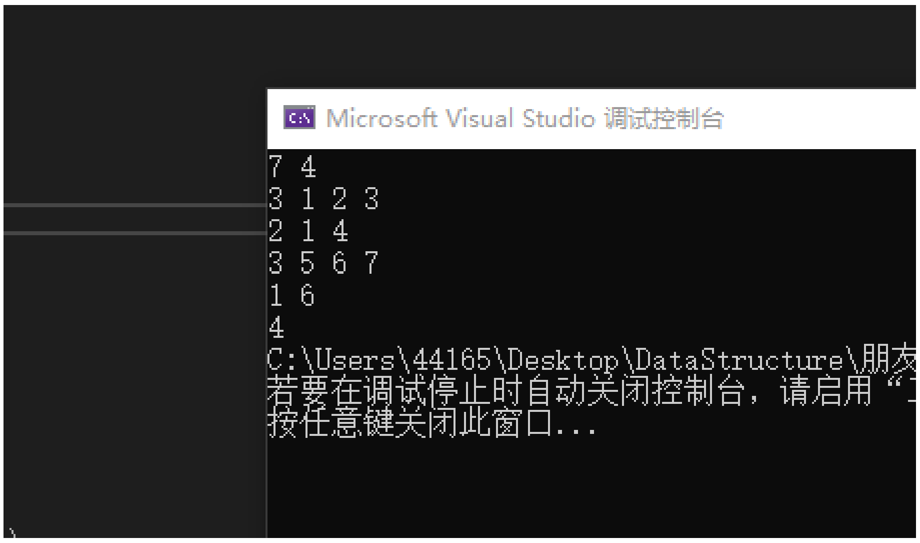


提交时间	状态	分数	题目	编译器	耗时	用户
2020/1/8 09:45:56	答案正确	25	7-25	C++ (g++)	46 ms	Cccc

测试点	提示	结果	耗时	内存
0	sample	答案正确	4 ms	384 KB
1	最小M；有学生没加入任何俱乐部	答案正确	4 ms	384 KB
2	最大M和N，随机	答案正确	46 ms	640 KB

代码

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
int fx[30001];
int sum[30001];
```



Microsoft Visual Studio 调试控制台

```
7 4
3 1 2 3
2 1 4
3 5 6 7
1 6
4
C:\Users\44165\Desktop\DataStructure\朋友
若要在调试停止时自动关闭控制台，请启用 “
按任意键关闭此窗口. . .
```

4. 算法时间复杂度及优化

单纯的遍历题，时间复杂度为 O(n)。

无优化。

5. 源代码

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
int fx[30001];
int sum[30001];
int Find(int x)
{
 if (fx[x] == x)
     return x;
 return Find(fx[x]);
```

```cpp
}

void add(int x,int y)
{
    int a = Find(x);
    int b = Find(y);
    if (a != b)
    {
        fx[b] = a;
    }
}

int main()
{
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        fx[i] = i;
    while (m--)
    {
        int num, x, y;
        cin >> num;
        for (int i = 1; i <= num; i++)
        {
            if (i == 1)
                cin >> x;
            else
            {
                cin >> y;
                add(x, y);
            }
        }
    }
    int ans = 0;
    for (int i = 1; i <= n; i++)
    {
        sum[Find(i)]++;
    }
    for (int i = 1; i <= n; i++)
    {
        ans = max(ans, sum[i]);
    }
    cout << ans;
}
```

11. 社交网络中的重要性计算

1. 数据结构

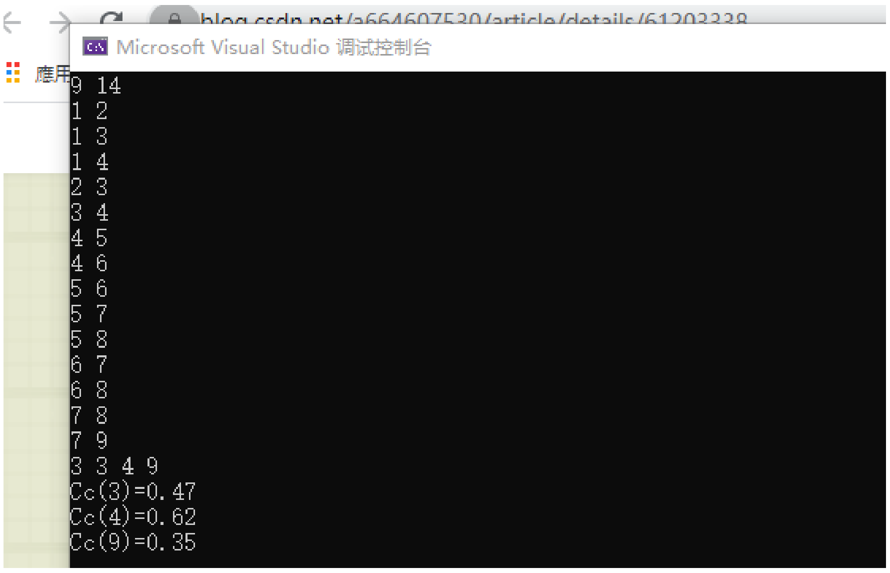
   邻接表

2. 设计思想

   先通过 BFS 判断是否为连通图，通过迪杰斯特拉计算每个点的最短路径

   然后直接计算

3. 样例及测试数据分析



```
9 14
1 2
1 3
1 4
2 3
3 4
4 5
4 6
5 6
5 7
5 8
6 7
6 8
7 8
7 9
3 3 4 9
Cc(3)=0.47
Cc(4)=0.62
Cc(9)=0.35
```

4. 算法时间复杂度及优化

   时间复杂度 O(n^2)，优化可以使用堆优化。

5. 源代码

```
#include<iostream>
#include<vector>
#include<iomanip>
#include<queue>
#define INT_MAX 500000
using namespace std;
struct Edge {
  vector<int> to;
  int name;
};
```

```cpp

struct Graph {
 vector<Edge> e;
 int vexnum, arcnum;
};

struct DjNode {
 int name;
 int distance;
 int pre;
 int flag;
};
DjNode* DJ = new DjNode[5000];
void Dj(int start, Graph G)
{
 for (int i = 0; i < G.vexnum; i++)
 {
     DJ[i].distance = INT_MAX - 1;
     DJ[i].flag = false;
     DJ[i].name = G.e[i].name;
     DJ[i].pre = start;
 }
 DJ[start - 1].pre = start;
 DJ[start - 1].flag = true;
 DJ[start - 1].distance = 0;
 for (int i = 0; i < G.e[start-1].to.size(); i++)
 {
     if (DJ[start-1].distance + 1 < DJ[G.e[start-1].to[i] - 1].distance)
     {
         DJ[G.e[start-1].to[i] - 1].distance = DJ[start-1].distance + 1;
         DJ[G.e[start-1].to[i] - 1].pre = start;
     }
 }
 while (1)
 {
     int minnposition = 0, minn = INT_MAX;
     for (int i = 0; i < G.vexnum; i++)
     {
         if (DJ[i].flag == true)
             continue;
         if (DJ[i].distance < minn)
         {
             minn = DJ[i].distance;
             minnposition = i;
```

```
            }
        }
        DJ[minnposition].flag = true;
        for (int i = 0; i < G.e[minnposition].to.size(); i++)
        {
            if (DJ[minnposition].distance + 1 < DJ[G.e[minnposition].to[i] - 1].distance)
            {
                DJ[G.e[minnposition].to[i] - 1].distance = DJ[minnposition].distance + 1;
                DJ[G.e[minnposition].to[i] - 1].pre = minnposition + 1;
            }
        }
        int sum = 0;
        for (int i = 0; i < G.vexnum; i++)
        {
            if (DJ[i].flag == true)
            {
                sum++;
            }
        }
        if (sum == G.vexnum)
            break;
    }
}
queue<int> s;
int Visit[100000];
int BFS(Graph G)
{
    int sum = 0;
    s.push(G.e[0].name);
    Visit[G.e[0].name] = 1;
    sum++;
    while (!s.empty())
    {
        int temp = s.front();
        s.pop();
        for (int i = 0; i < G.e[temp - 1].to.size(); i++)
        {
            if (Visit[G.e[temp - 1].to[i]] != 1)
            {
                s.push(G.e[temp - 1].to[i]);
                Visit[G.e[temp - 1].to[i]] = 1;
                sum++;
            }
        }
    }
    if (sum != G.vexnum)
    {
        return 0;
    }
    return 1;
}

int main()
{
    int n, m;
    Graph G;
    vector<int> nope;
    cin >> n >> m;
    for (int i = 0; i < n; i++)
    {
        G.e.push_back({ nope,i + 1 });
    }
    G.vexnum = n;
    G.arcnum = m;
    while (m--)
    {
        int start, end;
        cin >> start >> end;
        G.e[start - 1].to.push_back(end);
        G.e[end - 1].to.push_back(start);
    }
    int t, s;
    int g = BFS(G);
    cin >> t;
    while (t--)
    {
        cin >> s;
        Dj(s,G);
        float ssum = 0;
        float end;
        for (int i = 0; i < G.vexnum; i++)
        {
            if (i == s - 1)
                continue;
            if (DJ[i].distance == INT_MAX - 1)
            {
                continue;
            }
            ssum += DJ[i].distance;
        }
        if (g == 1)
        {
            end = 1 / (ssum / (G.vexnum - 1));
            cout << "Cc(" << s << ")=" << setprecision(2) << end << endl;
        }
        else if (g == 0)
        {
            cout << "Cc(" << s << ")=" << 0 << endl;
        }
    }
}
```

12. 行车路线

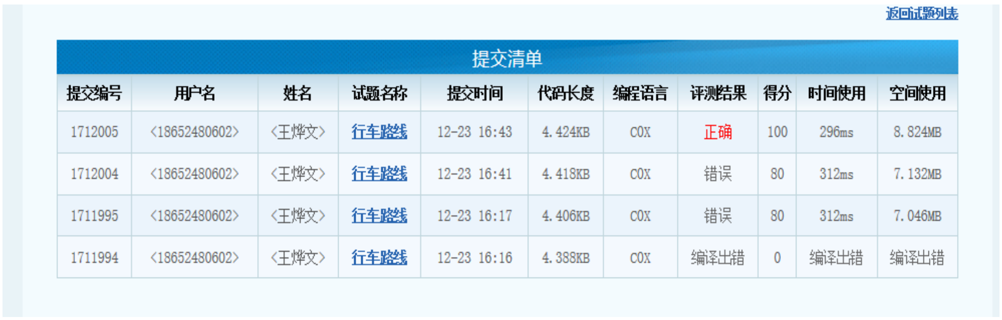
1. 数据结构

   邻接表

2. 设计思想

   通过改变迪杰斯特拉算法，将距离改变为疲劳度，注意要暂存前面的平方结果，如果继续走小路，则需要加上前面已经叠加的平方基数在进行平方相加。值得注意的是这里的数据类型最好为 long long，不然无法通过。

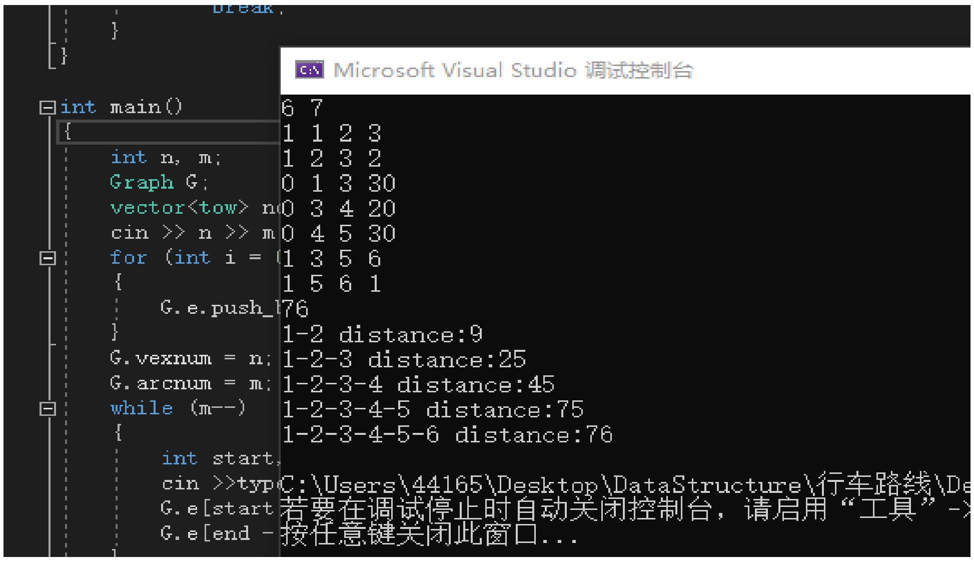
3. 样例及测试数据分析



返回试题列表

提交清单

提交编号	用户名	姓名	试题名称	提交时间	代码长度	编程语言	评测结果	得分	时间使用	空间使用
1712005	<18652480602>	<王烨文>	行车路线	12-23 16:43	4.424KB	C0X	正确	100	296ms	8.824MB
1712004	<18652480602>	<王烨文>	行车路线	12-23 16:41	4.418KB	C0X	错误	80	312ms	7.132MB
1711995	<18652480602>	<王烨文>	行车路线	12-23 16:17	4.406KB	C0X	错误	80	312ms	7.046MB
1711994	<18652480602>	<王烨文>	行车路线	12-23 16:16	4.388KB	C0X	编译出错	0	编译出错	编译出错



```
            break;
        }
}

int main()
{
    int n, m;
    Graph G;
    vector<tow> n
    cin >> n >> m
    for (int i = (
    {
        G.e.push_
    }
    G.vexnum = n;
    G.arcnum = m;
    while (m--)
    {
        int start,
        cin >>typ
        G.e[start
        G.e[end -
```

Microsoft Visual Studio 调试控制台

```
6 7
1 1 2 3
1 2 3 2
0 1 3 30
0 3 4 20
0 4 5 30
1 3 5 6
1 5 6 1
76
1-2 distance:9
1-2-3 distance:25
1-2-3-4 distance:45
1-2-3-4-5 distance:75
1-2-3-4-5-6 distance:76

C:\Users\44165\Desktop\DataStructure\行车路线\De
若要在调试停止时自动关闭控制台，请启用“工具”->
按任意键关闭此窗口...
```

4. 算法时间复杂度及优化

时间复杂度：迪杰斯特拉：O(n^2)。

优化：堆优化。

5. 源代码

```
#include<iostream>
#include<vector>
#include<math.h>
#include<iomanip>
using namespace std;
struct tow {
  int name;
  int type;
  long long distance;
};
struct Edge {
  vector<tow> to;
  int name;
};

struct Graph {
  vector<Edge> e;
  int vexnum, arcnum;
};

struct DjNode {
  int name;
```

```
 long long distance;
 int pre;
 int flag;
 int type;
 long long sroad;
};
DjNode* DJ = new DjNode[5000];
void Dj(int start, Graph G)
{
 for (int i = 0; i < G.vexnum; i++)
 {
     DJ[i].distance = INT_MAX - 1;
     DJ[i].flag = false;
     DJ[i].name = G.e[i].name;
     DJ[i].pre = start;
     DJ[i].type = 0;
     DJ[i].sroad = 0;
 }
 DJ[start - 1].pre = start;
 DJ[start - 1].flag = true;
 DJ[start - 1].distance = 0;
 for (int i = 0; i < G.e[start - 1].to.size(); i++)
 {
     if (G.e[start - 1].to[i].type == 0)
     {
         if (DJ[start - 1].distance + G.e[start-1].to[i].distance < DJ[G.e[start - 1].to[i].name - 1].distance)
         {
             DJ[G.e[start - 1].to[i].name - 1].distance = DJ[start - 1].distance + G.e[start - 1].to[i].distance;
             DJ[G.e[start - 1].to[i].name - 1].pre = start;
             DJ[G.e[start - 1].to[i].name - 1].type = 0;
             DJ[G.e[start - 1].to[i].name - 1].sroad = 0;
         }
     }
     else
     {
         if (DJ[start - 1].distance + G.e[start - 1].to[i].distance*G.e[start-1].to[i].distance < DJ[G.e[start - 1].to[i].name - 1].distance)
         {
             DJ[G.e[start - 1].to[i].name - 1].distance = DJ[start - 1].distance + G.e[start - 1].to[i].distance * G.e[start - 1].to[i].distance;
             DJ[G.e[start - 1].to[i].name - 1].pre = start;
             DJ[G.e[start - 1].to[i].name - 1].type = 1;
```

```
                DJ[G.e[start - 1].to[i].name - 1].sroad = G.e[start - 1].to[i].distance * G.e[start - 1].to[i].distance;
            }
        }
    }
    while (1)
    {
        int minnposition = 0, minn = INT_MAX;
        for (int i = 0; i < G.vexnum; i++)
        {
            if (DJ[i].flag == true)
                continue;
            if (DJ[i].distance < minn)
            {
                minn = DJ[i].distance;
                minnposition = i;
            }
        }
        DJ[minnposition].flag = true;
        for (int i = 0; i < G.e[minnposition].to.size(); i++)
        {
            if (G.e[minnposition].to[i].type == 0)
            {
                if (DJ[minnposition].distance + G.e[minnposition].to[i].distance < DJ[G.e[minnposition].to[i].name - 1].distance)
                {
                    DJ[G.e[minnposition].to[i].name - 1].distance = DJ[minnposition].distance + G.e[minnposition].to[i].distance;
                    DJ[G.e[minnposition].to[i].name - 1].pre = minnposition+1;
                    DJ[G.e[minnposition].to[i].name - 1].sroad = 0;
                    DJ[G.e[minnposition].to[i].name - 1].type = 0;
                }
            }
            else if(G.e[minnposition].to[i].type==1&&DJ[minnposition].type==0)
            {
                if (DJ[minnposition].distance + G.e[minnposition].to[i].distance * G.e[minnposition].to[i].distance < DJ[G.e[minnposition].to[i].name - 1].distance)
                {
                    DJ[G.e[minnposition].to[i].name - 1].distance = DJ[minnposition].distance + G.e[minnposition].to[i].distance * G.e[minnposition].to[i].distance;
                    DJ[G.e[minnposition].to[i].name - 1].pre = minnposition+1;
                    DJ[G.e[minnposition].to[i].name - 1].sroad =
```

```cpp
G.e[minnposition].to[i].distance * G.e[minnposition].to[i].distance;
                    DJ[G.e[minnposition].to[i].name - 1].type = 1;
                }
            }
            else if (G.e[minnposition].to[i].type == 1 && DJ[minnposition].type == 1)
            {
                if (DJ[minnposition].distance-DJ[minnposition].sroad + (sqrt(DJ[minnposition].sroad) + G.e[minnposition].to[i].distance) * (sqrt(DJ[minnposition].sroad) + G.e[minnposition].to[i].distance) < DJ[G.e[minnposition].to[i].name - 1].distance)
                {
                    DJ[G.e[minnposition].to[i].name - 1].distance = DJ[minnposition].distance-DJ[minnposition].sroad + (sqrt(DJ[minnposition].sroad) + G.e[minnposition].to[i].distance) * (sqrt(DJ[minnposition].sroad) + G.e[minnposition].to[i].distance);
                    DJ[G.e[minnposition].to[i].name - 1].pre = minnposition + 1;
                    DJ[G.e[minnposition].to[i].name - 1].sroad = (sqrt(DJ[minnposition].sroad) + G.e[minnposition].to[i].distance) * (sqrt(DJ[minnposition].sroad) + G.e[minnposition].to[i].distance);
                    DJ[G.e[minnposition].to[i].name - 1].type = 1;
                }
            }
        }
        int sum = 0;
        for (int i = 0; i < G.vexnum; i++)
        {
            if (DJ[i].flag == true)
            {
                sum++;
            }
        }
        if (sum == G.vexnum)
            break;
    }
}

int main()
{
    int n, m;
    Graph G;
    vector<tow> nope;
    cin >> n >> m;
    for (int i = 0; i < n; i++)
    {
        G.e.push_back({ nope,i + 1 });
    }
    G.vexnum = n;
    G.arcnum = m;
    while (m--)
    {
        int start, end,type,distance;
        cin >>type>> start >> end>>distance;
        G.e[start - 1].to.push_back({end,type,distance});
        G.e[end - 1].to.push_back({start,type,distance});
    }
    Dj(1,G);
    cout << DJ[G.vexnum - 1].distance << endl;
    vector<int> sta;
    for (int i = 1; i < G.vexnum; i++)
    {
        int temp = DJ[i].name;
        while (DJ[temp - 1].pre != temp)
        {
            temp = DJ[temp - 1].pre;
            sta.push_back(temp);
        }
        for (int i = sta.size() - 1; i >= 0; i--)
        {
            cout << sta[i] << "-";
        }
        cout << DJ[i].name << " distance:" << DJ[i].distance << endl;
        sta.clear();
    }
}
```

13. 跳一跳

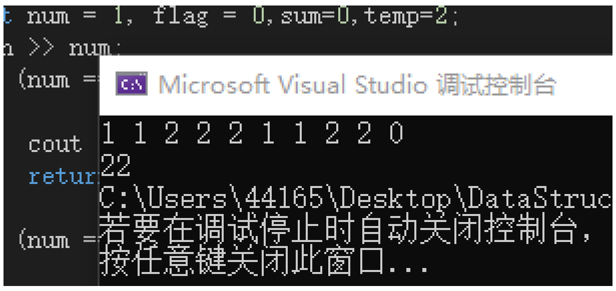
1. 数据结构

   数组。

2. 设计思想

   纯模拟，一步一步操作。

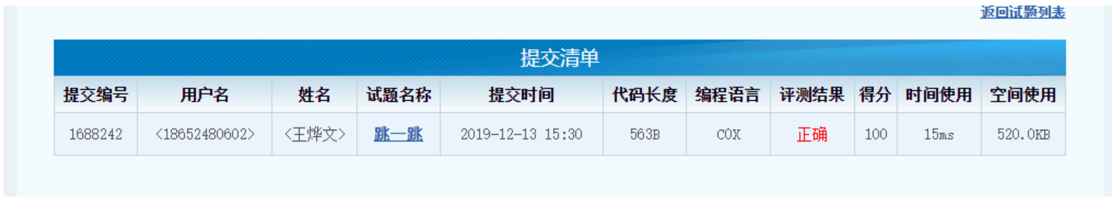
3. 样例及测试数据分析



```
t num = 1, flag = 0,sum=0,temp=2;
n >> num;
 (num =
  cout
  retur
 (num =
```

Microsoft Visual Studio 调试控制台

```
1 1 2 2 2 1 1 2 2 0
22
C:\Users\44165\Desktop\DataStruc
若要在调试停止时自动关闭控制台，
按任意键关闭此窗口. . .
```



返回试题列表

提交清单

提交编号	用户名	姓名	试题名称	提交时间	代码长度	编程语言	评测结果	得分	时间使用	空间使用
1688242	<18652480602>	<王烨文>	跳一跳	2019-12-13 15:30	563B	C0X	正确	100	15ms	520.0KB

4. 时间复杂度及优化

时间复杂度O(n)。无优化，纯模拟。

5. 源代码

```cpp
#include<iostream>
using namespace std;
int main()
{
 int num = 1, flag = 0,sum=0,temp=2;
 cin >> num;
 if (num == 0)
 {
      cout << 0 << endl;
      return 0;
 }
 if (num == 1)
 {
      sum += 1;

 }
 if (num == 2)
 {
      sum += 2;
      flag = 1;
      temp += 2;
 }
 while (num != 0)
 {
      cin >> num;
      if (num == 0)
```

```
            break;
        else if (num == 1)
        {
            flag = 0;
            sum += 1;
            temp = 2;
        }
        else if (num == 2 && flag == 0)
        {
            sum += 2;
            flag = 1;
            temp += 2;
        }
        else if (num == 2 && flag == 1)
        {
            sum += temp;
            flag = 1;
            temp += 2;
        }
    }
    cout << sum;
}
```

14. 迷宫问题

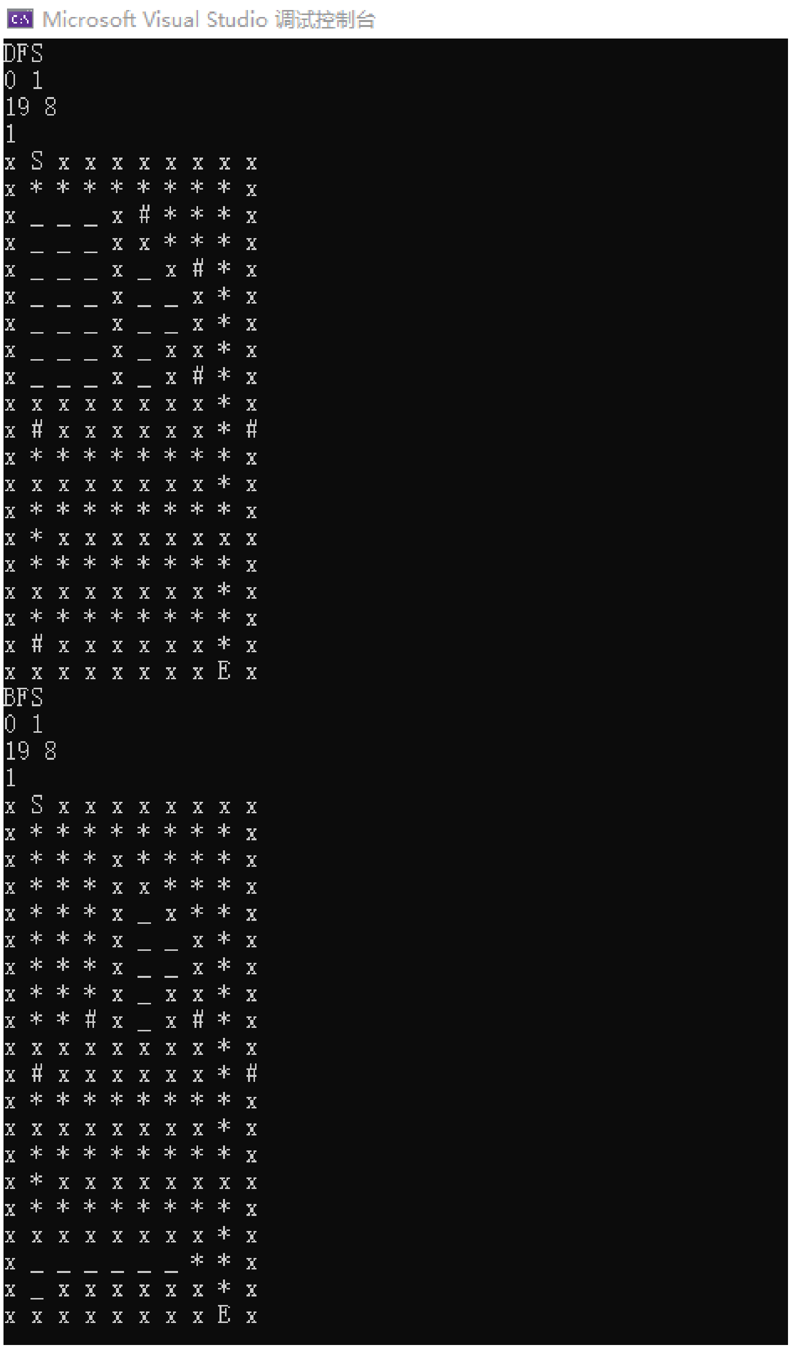
1. 数据结构

   堆栈，队列。

2. 设计思想

   BFS通过先进队起始点，然后进队其相邻的可走点，并标记已走，及数组值为1，然后再出队进行相同操作，直到队空。DFS则是采用堆栈进行相同操作。

3. 样例及测试数据分析


Microsoft Visual Studio 调试控制台

```
DFS
0 1
19 8
1
x S x x x x x x x x
x * * * * * * * * x
x _ _ _ x # * * * x
x _ _ _ x x * * * x
x _ _ _ x _ x # * x
x _ _ _ x _ _ x * x
x _ _ _ x _ _ x * x
x _ _ _ x _ x x * x
x _ _ _ x _ x # * x
x x x x x x x x * x
x # x x x x x x * #
x * * * * * * * * x
x x x x x x x x * x
x * * * * * * * * x
x * x x x x x x x x
x * * * * * * * * x
x x x x x x x x * x
x * * * * * * * * x
x # x x x x x x * x
x x x x x x x x E x
BFS
0 1
19 8
1
x S x x x x x x x x
x * * * * * * * * x
x * * * x * * * * x
x * * * x x * * * x
x * * * x _ x * * x
x * * * x _ _ x * x
x * * * x _ _ x * x
x * * * x _ x x * x
x * * # x _ x # * x
x x x x x x x x * x
x # x x x x x x * #
x * * * * * * * * x
x x x x x x x x * x
x * * * * * * * * x
x * x x x x x x x x
x * * * * * * * * x
x x x x x x x x * x
x _ _ _ _ _ _ * * x
x _ x x x x x x * x
x x x x x x x x E x
```

可以看见广度搜索可用于查找最短路径，而深度搜索可以用于查找多条路径。

4. 时间复杂度及优化

   BFS与DFS都是对点的遍历，所以时间复杂度取决于点的个数，O(n)。

   优化则是写成非递归形式。

5. 源代码

```cpp
#include<iostream>
#include<stack>
#include<memory>
#include<queue>
#include<fstream>
using namespace std;
int sx, sy, ex, ey;
int n, m;
int Graph[1000][1000];
bool flag[1000][1000];
int dx[4] = { 1,0,-1,0 };
int dy[4] = { 0,1,0,-1 };
struct node {
 int x=sx;
 int y=sy;
};
stack<node> s;
queue<node> path;
int BFS()
{
 node start;
 path.push(start);
 while (!path.empty())
 {
     int flag1 = 0;
     node start = path.front();
     path.pop();
     flag[start.x][start.y] = true;
     if (start.x == ex && start.y == ey)
     {
         return 1;
     }
     Graph[start.x][start.y] = 2;
     for (int i = 0; i < 4; i++)
     {
         if (start.x + dx[i] >= 0 && start.x + dx[i] <= n - 1 && start.y +
dy[i] >= 0 && start.y + dy[i] <= m -
1&&flag[start.x+dx[i]][start.y+dy[i]]==false)
         {
             node temp;
             temp.x = start.x + dx[i];
             temp.y = start.y + dy[i];
             path.push(temp);
             flag1 = 1;
         }
     }
     if (flag1 == 0)
     {
         Graph[start.x][start.y] = 3;
     }
 }
 return 0;
}

int DFS()
{
 node start;
 s.push(start);
 while (!s.empty())
 {
     int flag1 = 0;
     node start = s.top();
     s.pop();
     flag[start.x][start.y] = true;
     if (start.x == ex && start.y == ey)
     {
         return 1;
     }
     Graph[start.x][start.y] = 2;
     for (int i = 0; i < 4; i++)
     {
         if (start.x + dx[i] >= 0 && start.x + dx[i] <= n - 1 && start.y + dy[i] >= 0 && start.y + dy[i] <= m - 1 && flag[start.x + dx[i]][start.y + dy[i]] == false)
         {
             node temp;
             temp.x = start.x + dx[i];
             temp.y = start.y + dy[i];
             s.push(temp);
             flag1 = 1;
         }
     }
     if (flag1 == 0)
     {
         Graph[start.x][start.y] = 3;
     }
 }
 return 0;
```

```
}

int main()
{
    fstream fp;
    memset(flag, false, sizeof(flag));
    cout << "DFS" << endl;
    fp.open("1.txt", ios::in);
    fp >> n >> m;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            fp >> Graph[i][j];
            if (Graph[i][j] == 1)
                    flag[i][j] = true;
            }
        }
        cin >> sx >> sy >> ex >> ey;
        cout << DFS()<<endl;
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < m; j++)
            {
                if (Graph[i][j] == 2 && i == sx && j == sy)
                {
                    cout << "S ";
                    continue;
                }
                if (Graph[i][j] == 0 && i == ex && j == ey)
                {
                    cout << "E ";
                    continue;
                }
                if (Graph[i][j] == 0)
                {
                    cout << "_ ";
                }
                if (Graph[i][j] == 1)
                {
                    cout << "x ";
                }
                if (Graph[i][j] == 2)
                {
```

```
                cout << "* ";
            }
            if (Graph[i][j] == 3)
            {
                cout << "# ";
            }
        }
        cout << endl;
    }
    fp.close();
    memset(flag, false, sizeof(flag));
    cout << "BFS" << endl;
    fp.open("1.txt", ios::in);
    fp >> n >> m;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            fp >> Graph[i][j];
            if (Graph[i][j] == 1)
                flag[i][j] = true;
        }
    }
    cin >> sx >> sy >> ex >> ey;
    cout << BFS() << endl;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            if (Graph[i][j] == 2 && i == sx && j == sy)
            {
                cout << "S ";
                continue;
            }
            if (Graph[i][j] == 0 && i == ex && j == ey)
            {
                cout << "E ";
                continue;
            }
            if (Graph[i][j] == 0)
            {
                cout << "_ ";
            }
            if (Graph[i][j] == 1)
            {
                cout << "x ";
            }
            if (Graph[i][j] == 2)
            {
                cout << "* ";
            }
            if (Graph[i][j] == 3)
            {
                cout << "# ";
            }
        }
        cout << endl;
    }
    fp.close();
}
```

15. 键树

1. 数据结构

   孩子兄弟链表树。

2. 设计思想

   ① 查找，按照字母顺序查找单词，找到则比较下一层和下一位。

   ② 插入，一直查找找到字母开头单词与键树已存在单词相同处，然后在找到的地方插入剩下的字符。

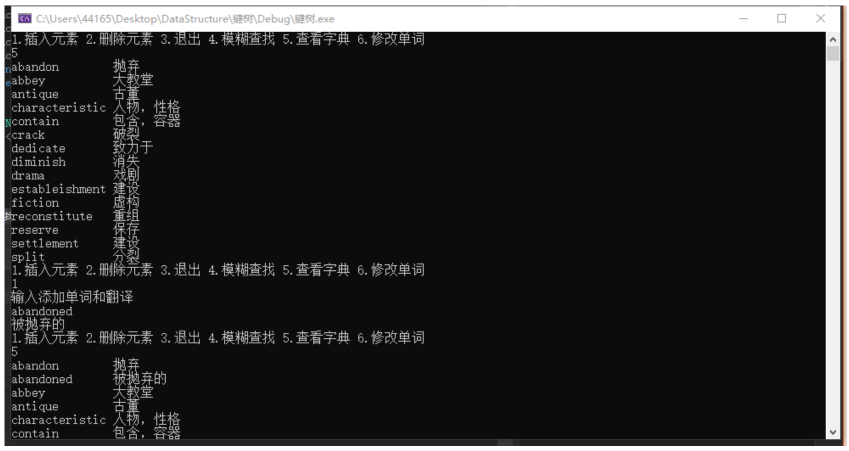
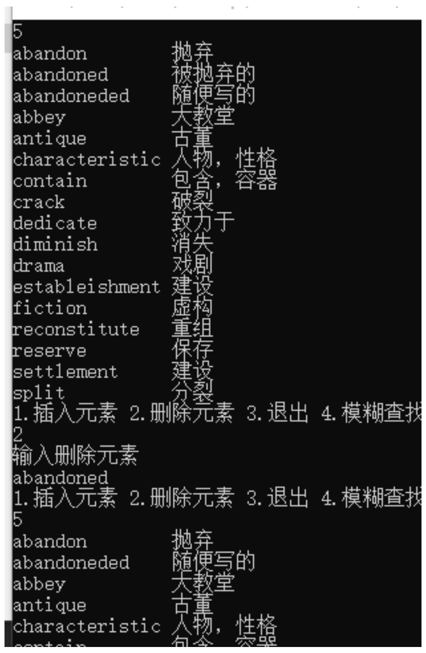
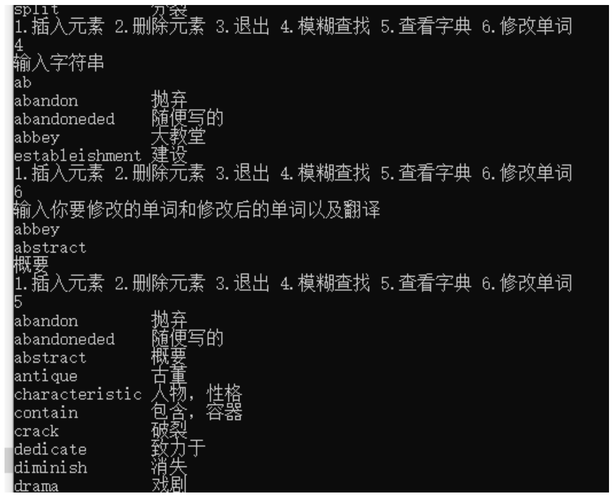
   ③ 删除，查找并将所有结点放入堆栈，然后从下往上查看，先删除这个单词的后缀‘$’，之后的节点，如果该节点存在兄弟节点或孩子节点，则不删除，不存在则删除。

   ④ 修改单词，由于单词的改动比较大，直接采取删除后再添加的方法。

   ⑤ 遍历：先序遍历所有字符并将其存入的string中，回退的时候在弹出一个字符，遇到‘$’，存入队列并按照字典序排序，然后调用map中

与单词对应的翻译进行展示。

3. 样例及测试数据分析

添加	
删除	
修改	

```
C:\Users\44165\Desktop\DataStructure\键树\Debug\键树.exe
1.插入元素 2.删除元素 3.退出 4.模糊查找 5.查看字典 6.修改单词
5
abandon       抛弃
abbey         大教堂
antique       古董
characteristic 人物，性格
contain       包含，容器
crack         破裂
dedicate      致力于
diminish      消失
drama         戏剧
estableishment 建设
fiction       虚构
reconstitute  重组
reserve       保存
settlement    建设
split         分裂
1.插入元素 2.删除元素 3.退出 4.模糊查找 5.查看字典 6.修改单词
1
输入添加单词和翻译
abandoned
被抛弃的
1.插入元素 2.删除元素 3.退出 4.模糊查找 5.查看字典 6.修改单词
5
abandon       抛弃
abandoned     被抛弃的
abbey         大教堂
antique       古董
characteristic 人物，性格
contain       包含，容器
```

```
5
abandon       抛弃
abandoned     被抛弃的
abandoneded   随便写的
abbey         大教堂
antique       古董
characteristic 人物，性格
contain       包含，容器
crack         破裂
dedicate      致力于
diminish      消失
drama         戏剧
estableishment 建设
fiction       虚构
reconstitute  重组
reserve       保存
settlement    建设
split         分裂
1.插入元素 2.删除元素 3.退出 4.模糊查找
2
输入删除元素
abandoned
1.插入元素 2.删除元素 3.退出 4.模糊查找
5
abandon       抛弃
abandoneded   随便写的
abbey         大教堂
antique       古董
characteristic 人物，性格
contain       包含，容器
```

```
split         分裂
1.插入元素 2.删除元素 3.退出 4.模糊查找 5.查看字典 6.修改单词
4
输入字符串
ab
abandon       抛弃
abandoneded   随便写的
abbey         大教堂
estableishment 建设
1.插入元素 2.删除元素 3.退出 4.模糊查找 5.查看字典 6.修改单词
6
输入你要修改的单词和修改后的单词以及翻译
abbey
abstract
概要
1.插入元素 2.删除元素 3.退出 4.模糊查找 5.查看字典 6.修改单词
5
abandon       抛弃
abandoneded   随便写的
abstract      概要
antique       古董
characteristic 人物，性格
contain       包含，容器
crack         破裂
dedicate      致力于
diminish      消失
drama         戏剧
```

4. 算法时间复杂度及优化

① 查找：时间复杂度为遍历节点时间，O(n)。

② 插入删除修改遍历均为O(n)。

这是一个查找表，所以数据复杂度为O(n)。

优化：存储的时候即存为字典序，可以手动定义$与单词的关系，将其放在兄弟链表中的第一位。

5. 源代码

```
#include<iostream>
#include<stdio.h>
#include<stdlib.h>
#include<string>
#include<queue>
#include<fstream>
#include<stack>
#include<map>
#include<vector>
#include<iomanip>
using namespace std;
typedef struct CSNode {
 char data;
 struct CSNode* firstchild, * nextsilbing;
}CSNode, * CSTree;
map<string, string> mp;
int InitCSTree(CSTree& T)
{
 T = (CSNode*)malloc(sizeof(struct CSNode));
 T->data = '#';
 T->firstchild = NULL;
 T->nextsilbing = NULL;
 return 1;
}

int InsertString(CSTree& T, string a)
{
 CSNode* child, * silb;
 CSNode* temp = (CSNode*)malloc(sizeof(struct CSNode));
 if (T == NULL)
 {
     InitCSTree(T);
```

```
    child = T;
    InitCSTree(child->firstchild);
    child = child->firstchild;
    for (int i = 0; i < a.size(); i++)
    {
        child->data = a[i];
        InitCSTree(child->firstchild);
        child = child->firstchild;
    }
    child->data = '$';
    return 1;
}
else
{
    temp = T->firstchild;
    int pos = 0, flag = 0;
    while (temp)
    {
        if (temp->data == a[pos])
        {
            flag = 0;
            if (!temp->firstchild)
                break;
            temp = temp->firstchild;
            pos++;
            continue;
        }
        else
        {
            if (pos >= a.size())
                break;
            if (temp->data != a[pos])
                flag = 1;
            if (temp->nextsilbing == NULL)
                break;
        }
        temp = temp->nextsilbing;
    }
    if (pos >= a.size())
    {
        silb = (CSNode*)malloc(sizeof(CSNode));
        silb->data = '$';
        silb->firstchild = NULL;
        temp->nextsilbing = silb;
```

```cpp
        silb->nextsilbing = NULL;
        return 1;
    }
    if (!temp->nextsilbing && flag == 1)
    {
        silb = (CSNode*)malloc(sizeof(CSNode));
        silb->data = a[pos];
        silb->nextsilbing = NULL;
        temp->nextsilbing = silb;
        InitCSTree(silb->firstchild);
        silb = silb->firstchild;
        for (int i = pos + 1; i < a.size(); i++)
        {
            silb->data = a[i];
            InitCSTree(silb->firstchild);
            silb = silb->firstchild;
        }
        silb->data = '$';
    }
    if (flag == 0)
    {
        if (pos == a.size())
            cout << "该单词已存在" << endl;
            else
            {
                for (int i = pos + 1; i < a.size(); i++)
                {
                    InitCSTree(temp->firstchild);
                    temp = temp->firstchild;
                    temp->data = a[i];
                }
                InitCSTree(temp->firstchild);
                temp = temp->firstchild;
                temp->data = '$';
            }
        }
    }
}

int DeleteString(CSTree& T, string a)
{
    stack<CSNode*> ss;
    CSNode* temp = T;
    CSNode* temp2 = (CSNode*)malloc(sizeof(CSNode));
    CSNode* temp3 = (CSNode*)malloc(sizeof(CSNode));
    if (T == NULL)
    {
        cout << "字典不存在" << endl;
        return 0;
    }
    ss.push(T);
    temp = T->firstchild;
    int pos = 0, flag = 0;
    while (temp)
    {
        if (temp->data == a[pos])
        {
            flag = 0;
            if (!temp->firstchild)
                break;
            ss.push(temp);
            temp = temp->firstchild;
            pos++;
            continue;
        }
        temp = temp->nextsilbing;
    }
    int i = 0;
    while (ss.size() != 1)
    {
        temp2 = ss.top();
        ss.pop();
        temp3 = ss.top();
        if (temp2->firstchild && i == 0)
        {
            CSNode* temp4 = (CSNode*)malloc(sizeof(CSNode));
            temp4 = temp2->firstchild;
            if (temp4->data == '$')
            {
                temp2->firstchild = temp4->nextsilbing;
            }
            else
            {
                while (temp4->nextsilbing->data != '$')
                    temp4 = temp4->nextsilbing;
                CSNode* temp5 = temp4->nextsilbing;
                temp4->nextsilbing = temp5->nextsilbing;
                free(temp5);
            }
            i++;
            return 1;
        }
        CSNode* temp4 = (CSNode*)malloc(sizeof(CSNode));
        temp4 = temp3->firstchild;
        if (temp4 == temp2)
        {
            temp3->firstchild = temp2->nextsilbing;
            temp2 = NULL;
            continue;
        }
        while (temp4->nextsilbing != temp2)
        {
            temp4 = temp4->nextsilbing;
        }
        temp4->nextsilbing = temp2->nextsilbing;
        temp2 = NULL;
        i++;
    }
    return 1;
}

struct cmp {
    bool operator()(string x, string y)
    {
        return x > y;
    }
};
priority_queue<string,vector<string>,cmp> s;
stack<CSNode*> n;
void Traverse(CSTree T)
{
    string temp;
    CSNode* t = T;
    t = t->firstchild;
    while (t || !n.empty())
    {
        while (t)
        {
            if (t->data == '$')
            {
                n.push(t);
                temp.push_back(t->data);
```

```cpp
                s.push(temp);
                t = t->firstchild;
            }
            if (t == NULL)
                break;
            temp.push_back(t->data);
            n.push(t);
            t = t->firstchild;
            if (t == NULL)
                break;
            if (t->data == '$')
            {
                n.push(t);
                temp.push_back(t->data);
                s.push(temp);
                t = t->firstchild;
            }
        }
        if (!s.empty())
        {
            t = n.top();
            n.pop();
            t = t->nextsilbing;
            temp.pop_back();
        }
    }
    while (!s.empty())
    {
        string out = s.top();
        s.pop();
        out.pop_back();
        cout << setw(15) << left << out << setw(15) << left << mp[out] << endl;
    }
    while (!s.empty())
        s.pop();
    while (!n.empty())
        n.pop();
}

void Search(CSTree T, string a)
{
    string temp;
    CSNode* t = T;
    t = t->firstchild;
    while (t || !n.empty())
    {
        while (t)
        {
            if (t->data == '$')
            {
                n.push(t);
                temp.push_back(t->data);
                s.push(temp);
                t = t->firstchild;
            }
            if (t == NULL)
                break;
            temp.push_back(t->data);
            n.push(t);
            t = t->firstchild;
            if (t == NULL)
                break;
            if (t->data == '$')
            {
                n.push(t);
                temp.push_back(t->data);
                s.push(temp);
                t = t->firstchild;
            }
        }
        if (!s.empty())
        {
            t = n.top();
            n.pop();
            t = t->nextsilbing;
            temp.pop_back();
        }
    }
    while (!s.empty())
    {
        string out = s.top();
        s.pop();
        out.pop_back();
        string::size_type idx;
        idx = out.find(a);
        if (idx == string::npos)
            continue;
        else
            cout << setw(15) <<left<< out << setw(15) <<left<< mp[out] << endl;
    }
    while (!s.empty())
        s.pop();
    while (!n.empty())
        n.pop();
}

void CreateCSTree(CSTree& T)
{
    fstream fp;
    fp.open("1.txt", ios::in);
    while (!fp.eof())
    {
        string temp,trans;
        fp >> temp>>trans;
        int flag = 0;
        map<string, string>::iterator it;
        it = mp.begin();
        while (it != mp.end())
        {
            if (it->first == temp)
            {
                flag = 1;
                break;
            }
            it++;
        }
        if (flag == 1)
            continue;
        else
            InsertString(T, temp);
        mp.insert({ temp,trans });
    }
    fp.close();
}

void modify(CSTree& T, string a, string b)
{
    DeleteString(T, a);
    InsertString(T, b);
}
```

```

int exist(string a)
{
    map<string, string>::iterator it=mp.begin();
    while (it != mp.end())
    {
        if (it->first == a)
        {
            return 0;
        }
        it++;
    }
    return 1;
}

void menu(CSTree& T)
{
    while (1)
    {
        string a,trans;
        cout << "1.插入元素 " << "2.删除元素 " << "3.退出 " << "4.模糊查找 "
 << "5.查看字典 " <<"6.修改单词"<< endl;
        int choice;
        cin >> choice;
        if (choice == 1)
        {
            cout << "输入添加单词和翻译" << endl;
            cin >> a >> trans;
            if (exist(a) == 0)
            {
                cout << "该单词已存在" << endl;
            }
            else
            {
                InsertString(T, a);
                mp.insert({ a,trans });
            }
        }
        else if (choice == 2)
        {
            cout << "输入删除元素" << endl;
            cin >> a;
            if (exist(a) == 0)
            {
                DeleteString(T, a);
                mp.erase(a);
            }
            else
            {
                cout << "该单词不存在" << endl;
            }
        }
        else if (choice == 3)
        {
            break;
        }
        else if (choice == 4)
        {
            cout << "输入字符串" << endl;
            cin >> a;
            Search(T, a);
        }
        else if (choice == 5)
        {
            Traverse(T);
        }
        else if (choice == 6)
        {
            cout << "输入你要修改的单词和修改后的单词以及翻译" << endl;
            string mod, after;
            cin >> mod >> after >> trans;
            if (exist(mod) == 0)
            {
                modify(T, mod, after);
                mp.erase(mod);
                mp.insert({ after, trans });
            }
            else
            {
                cout << "该单词不存在" << endl;
            }
        }
    }
}

int main()
{
    CSTree T;
    T = NULL;
    CreateCSTree(T);
    menu(T);
}
```

二．课程设计总结

1.总体情况汇总

1. 代码量

    ① 进程查看：407行。

    ② 算术表达式求值：361行。

    ③ 公共钥匙盒：236行。

    ④ 家谱管理：930行。

    ⑤ Huffman树解码与编码：234行。

    ⑥ 最小生成树：282行。

    ⑦ 公交线路提示：337行。

    ⑧ 排序算法比较：667行。

    ⑨ 朋友圈：60行。

    ⑩ 社交网络：160行。

    ⑪ 行车路线：169行

    ⑫ 跳一跳：53行。

    ⑬ 迷宫问题：193行。

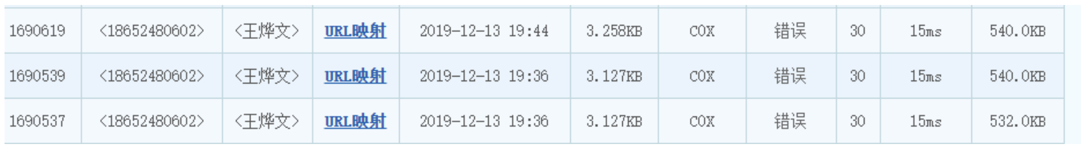
    ⑭ 键树：434行。

    总计4522行。

2.心得体会

数据结构课设是我在进入大学之后做的课设中遇到问题最多，请求他人帮

助最多，花费时间最多的一门课设。在一开始，特别是在写家谱树的时候，我那个时候没用使用Visual Studio，Develop对野指针基本是不怎么管理的，于是我用的很自然，但到了VS一堆的报错，以及学长的教育之后，我开始十分重视空间的申请和释放，然后我就再没写过野指针。本次课设帮我再一次明白的代码健壮性的重要性。同时，我也去网上查阅各种优化办法，并向人请教，比如说公交线路题，在一开始我的程序运行很慢，但我找不到原因，其实是时间复杂度+站点导致的时间过长，后来我朋友提议我去使用堆优化，于是我才发现了优先队列这个好东西，优化之后运行时间基本都在一秒之内。排序题，一开始自己写的是非递归模式，然后爆栈了，于是上网搜索了一下并学会了非递归写法，大部分代码都是通过非递归来完成。然后这次的课程设计也帮我重新回顾了一下kruskal算法和并查集的使用方法。在选做题中，我其实还选做了URL映射，但由于暴力匹配情况可能没有完全处理完全，只有30分。



1690619	<18652480602>	<王烨文>	URL映射	2019-12-13 19:44	3.258KB	COX	错误	30	15ms	540.0KB
1690539	<18652480602>	<王烨文>	URL映射	2019-12-13 19:36	3.127KB	COX	错误	30	15ms	540.0KB
1690537	<18652480602>	<王烨文>	URL映射	2019-12-13 19:36	3.127KB	COX	错误	30	15ms	532.0KB

同时，也自己尝试写了一下B-树，也算是拓宽了自己的知识面，红黑树的情况基本已经写出来了，但B-树没有写出来。

这次的数据结构课程设计也算是我一学期数据结构课程学习的成果，就和去年C++课程设计一样，我研究了更多的东西，拓宽了自己的知识面，也纠正了自己在平时边写代码时留下的坏习惯，也结交了许多朋友与大佬，也从他们那边学到了许多知识，他们也给了许多建议与提示。总之是受益匪浅，也让我的大二上学期在我的人生中留下了印记。